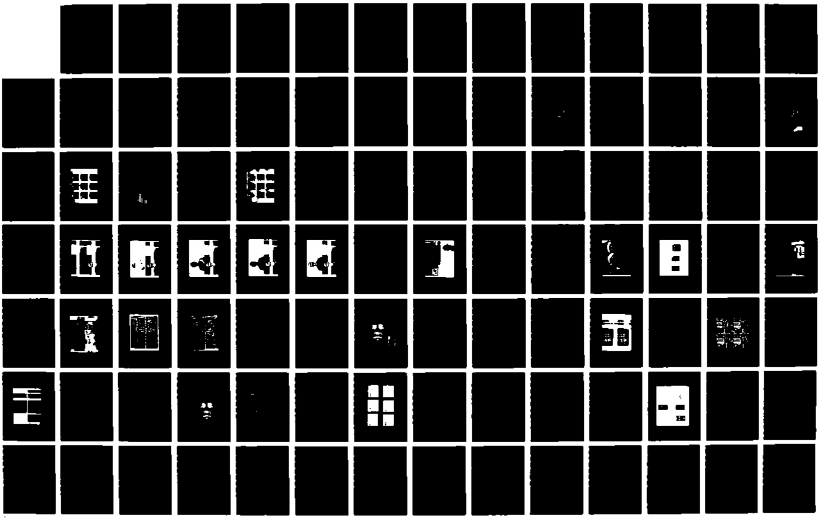
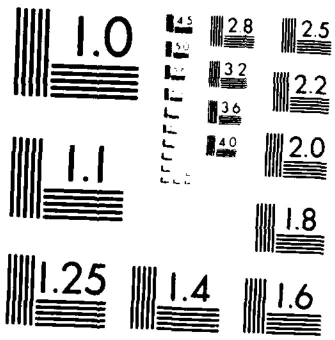
AD-A188 819 EVALUATION AND ENHANCEMENT OF THE AFIT AUTONOMOUS FACE 1/3
RECOGNITION MACHINE(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
UNCLASSIFIED L C LAMBERT DEC 87 AFIT/GE/ENG/87D-35 F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

Evaluation and Enhancement of the AFIT
Autonomous Face Recognition Machine

THESIS

Laurence C. Lambert
Captain, USAF

AFIT/GE/ENG/87D-35

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/87D-35

Evaluation and Enhancement of the AFIT
Autonomous Face Recognition Machine

THESIS

Laurence C. Lambert
Captain, USAF

AFIT/GE/ENG/87D-35

AFIT/GE/ENG/87D-35

Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Laurence C. Lambert, B.S.
Captain, USAF

December 1987

Acknowledgements

To Christopher Andrew Lambert. Over the past year I have poured my thoughts of you, my frustrations and my love for you, into this work in order to make it a fitting dedication to you. I thank God for the strength and endurance that I needed to complete this work and for the part that you have played and continue to play in my life.

To my wife Claire. You have shown more patience towards me than I ever deserved during this past year. But I guess that was easy for you because of all the years of experience you have had in dealing with me.

To Professor Matthew Kabrisky. Thankyou. Your advice, ideas, knowledge, humor, stories (in class these were often called hiatus), optimism and guidance have been invaluable.

To everyone else. Mr Dan Zambon, the student's best friend. Add 50% to your total manhours if you don't have Mr Zambon managing the equipment in your lab. My subjects, whose friendship I won't forget, nor will I forget their faces (they appear in various figures in this thesis).

Table of Contents

List of Figures

Figure Page

List of Tables

Abstract

This thesis evaluates and improves the Autonomous Face Recognition Machine (AFRM) created in 1985 at AFIT. This effort involved re-writing the AFRM code in the C programming language and hosting it on a Micro-VAX II. In addition, several new algorithms were added to the AFRM including: brightness normalization of input images, moving target detection, and a new face location algorithm. The results of this effort include: improved face location, higher recognition accuracy, and near real-time processing.

This thesis includes a complete description of the AFRM and its development history.

EVALUATION AND ENHANCEMENT OF THE AFIT AUTONOMOUS FACE RECOGNITION MACHINE

I. Introduction

Background

A Face Recognition Machine (FRM) was developed at AFIT in 1985 (Russel 1985). The FRM was based on Cortical Thought Theory (CTT) which proposes a new model of how a human brain processes information. Richard Routh developed and presented CTT as a doctoral dissertation at AFIT (Routh 1985). CTT proposes that information is displayed as a two-dimensional image on the brain. The brain then extracts the essential information (the essence of the image) as a two-dimensional vector, called a "gestalt". The gestalt is the only information that is passed to the higher levels of the brain for processing (Russel, 1985:3-1 to 3-2). The FRM reduces facial images to gestalts and then compares the gestalts to a database in an attempt to recognize the face.

In 1986 an AFIT student added automatic face location and windowing algorithms to the FRM to eliminate human influence on the recognition process (Smith, 1986). The face locator was slow and recognition was less accurate because only the internal features of the face (eyes, nose, mouth) were used, but the question this student was trying to answer was, "Can a machine, entirely on its own, determine whether or not a

person's face is in a picture and if so, can it determine to whom the face belongs?" (Smith, 1986:6-1). The answer is "Yes" and the result of the student's thesis effort became the Autonomous Face Recognition Machine (AFRM).

Problem Statement

This thesis effort evaluates the AFRM location and windowing algorithms with the goal of improving recognition score and speed. Both the score and speed were reduced with the addition of the autonomous scene analysis (location and windowing) algorithms in 1986, however human influence was eliminated. The goal of this effort was to reduce the 5 to 30 minute scene analysis time as much as possible while bringing the recognition score back up to at least what was possible when human influence was allowed.

Scope

Improvement of the windowing algorithms should improve overall recognition accuracy. There are several windows on the facial scene that will be tested as possible replacements for the windows that have little affect on the recognition score (Russel, 1985:6-11, 8-2). Going back to a whole-head approach used in 1985 should also improve accuracy over the internal feature approach now used (Smith,1986:6-2). The only reason internal features are now used is the inability to separate the edges of the head from a random background. This thesis investigates two possible solutions to this

problem. The first is to apply an elliptical mask to a scene centered around the location of the face with a size proportional to the size of the internal features. This results in a larger area of the face being made available to the recognition algorithm. The second solution is to apply a Moving Target Indicator (MTI) algorithm to a series of input scenes prior to scene analysis. This may allow better detection of the edge of the head.

Improvement of the location algorithm may speed up the scene analysis, however the major improvement in speed will be gained by re-hosting the AFRM on a new Micro-VAX in the Signal Processing Lab at AFIT.

Assumptions

The assumptions from the past efforts (Smith, 1986:1-5) that remain valid are as follows:

1. In any given picture the subject(s) are looking squarely at the camera (there is no tilt or rotation of the head).
2. The subjects are not wearing glasses and have relatively relaxed expressions (the face is not deliberately contorted).
3. Four pictures for each subject are sufficient to characterize a person in the database.
4. The basic CTT algorithm used in the AFRM is valid.

There are no assumptions about the contents of scenes fed to the AFRM. In order to be autonomous, the AFRM should be able to process a scene with a random background.

Standards

Test results must meet the same criteria as set out in the past thesis effort (Smith, 1986:1-5 to 1-6):

1. The system must demonstrate "human like" classification of human facial images.

2. Recognition performance must be as good as that obtained by Russel (Russel, 1985:6-9 to 6-13).

3. No operator intervention is allowed in the face location, windowing and recognition processes.

4. The system must be able to process pictures with multiple (at least two) faces in them.

Approach

The approach used was a top-down conversion of the software from the Data General computers to the new Micro-VAX computer. As software was re-hosted it was tested and compared to the original results. As algorithms were transferred, enhancements were made and tested.

In most cases the software had to be re-written using only the ideas from the original system because a different language was used and because the extensive communication requirements written for the two-computer configuration were no longer required.

When the AFRM was up and running on the Micro-VAX, the database was trained to test face recognition accuracy with the new facial windows and gestalt calculations.

Thesis Structure

Chapter 2 gives an overall description of the previous AFRM hosted on the two Data General computers. A review of some of the literature used to support the development of this AFRM is presented.

Chapter 3 evaluates the AFRM and describes enhancements made during this thesis effort. A review of the literature used to support the development of these enhancements is presented. Chapter 3 topics are in parallel with Chapter 2.

Chapter 4 describes how the algorithms that make up the AFRM have been gathered together into a complete program and implemented on the Micro-VAX. Although the structure of the AFRM is described, operational details have been left out as use of the AFRM is covered separately in the User's Manual appended to this thesis. The design of the database is also covered in detail in this chapter.

Chapter 5 presents the results of testing performed to verify the proper operation of the AFRM and to show the differences made by the various enhancements.

Chapter 6 contains conclusions based on test results and recommendations for further tests and enhancements.

II. Background of the AFIT Face Recognition System

This chapter presents an overview of the AFIT face recognition effort prior to this thesis effort. First there is a discussion of Cortical Thought Theory, and then a description of the face recognition system. Some parts of the face recognition process have been described in more detail than others so the reader will be prepared to evaluate the enhancements discussed in Chapter 3. The parts of the AFRM that are not affected by this thesis are only breifly discussed here. These areas are covered in detail in Russel's thesis (Russel, 1985).

Cortical Thought Theory

Cortical Thought Theory (CTT), developed by Capt Richard Routh, proposed a model of the human brain that was based on primitives of analogy as opposed to primitives of deduction. Routh described how primitives of analogy could be used to acheive human-like classification of data and human-like recall or, "direct memory access" (Routh, 1985:40-42). The classification, or single unique identification of an object, was called the "gestalt" of the object (Routh, 1985:2,3,39). The direct memory access ability comes from the technique of mapping gestalts onto the surface of the brain. The gestalt itself provides the location on the cortex that can identify the input data (Routh, 1985:96-97).

The scope of Capt Routh's dissertation was to find a gestalt mechanism that was reasonable for the brain to accomplish and that was in accordance with what was known about the neurophysiological structure of the brain (Routh 1985:3). As part of this dissertation, Routh demonstrated his gestalt mechanism by applying it to a speech recognition problem. Figure 2-1 shows how an audio input was transformed into a gestalt value that identifies the word being spoken. Overlapping time slices of the audio input are transformed into gestalt points on a phoneme mapping surface. These points taken together as a "phoneme track" are then transformed into a single gestalt point on the word mapping surface. This demonstration showed the "human-like" classification of inputs that Routh was looking for in a gestalt mechanism.

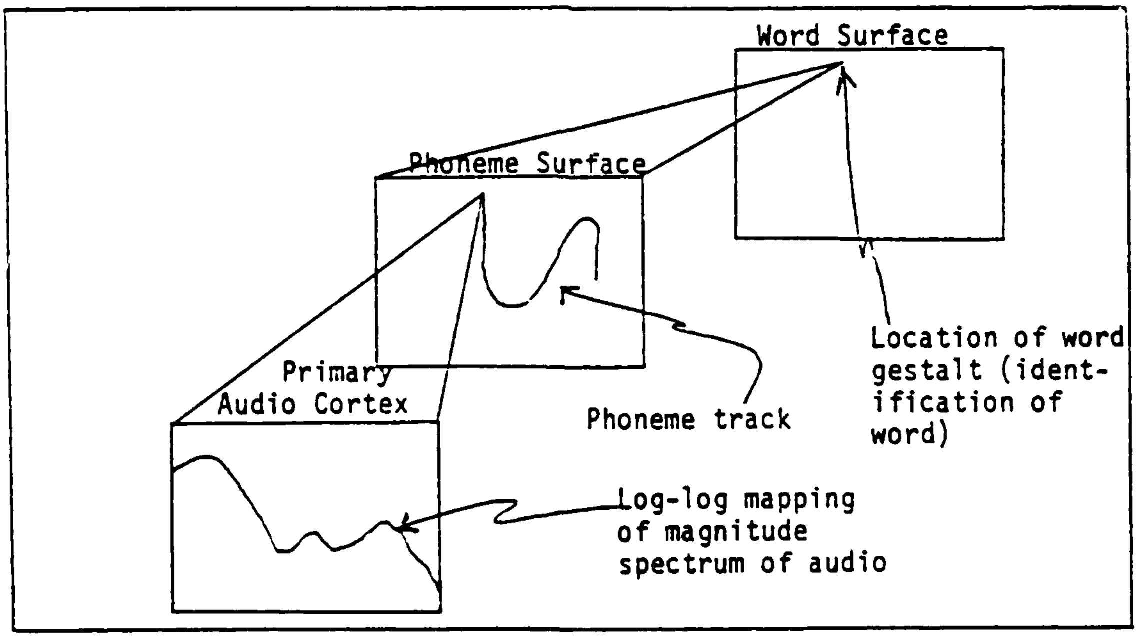


Figure 2-1 Application of CTT to Speech (Routh, 1985:156)

Figure 2-2 shows the separation of unlike inputs and the grouping of like inputs at the word mapping level. The speech recognizer also had the "human-like" recall ability required by CTT. The output of the recognizer was simply the closest word on the word mapping surface.

In 1985 Robert L. Russel applied CTT to the problem of face recognition (Russel, 1985:1-2). The results of Russel's work, "increases the credibility of CTT as a model of human sensory processing" (Russel, 1985:7-4). In 1986, Edward Smith added an automatic face location algorithm to Russel's face recognizer to make the recognition process independent of operator influence (Smith, 1985:1-4).

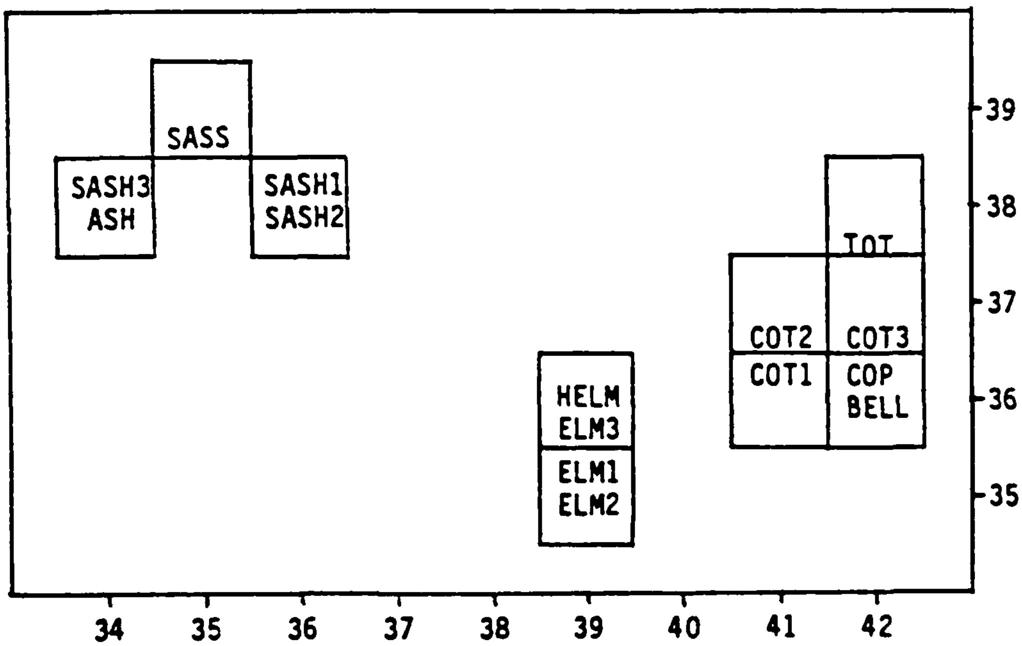


Figure 2-2 Gestalt Mapping for Words (Routh, 1985:168)

The Recognition System

The following sections describe the AFRM and how CTT is implemented in computer software.

System Environment

The AFRM was created using the two-computer configuration shown in Figure 2-3. The equipment is listed in Appendix A. The Nova computer was used for image acquisition and display, and the Eclipse was used for the large amount of numerical processing required by the gestalt calculations. The two computers shared a common disk drive and communicated via flag files stored on disk. In many cases these flag files existed in name only to tell one computer that a process was finished on the other. In some cases the files contained data that was to be passed from one computer to the other. Software for the recognizer was written in Fortran IV and Fortran V and extensive use of subroutine swapping and overlay techniques were employed due to the small main memory, approximately 28K bytes, available for running programs (Smith, 1986:4-1 to 4-4). Descriptions of the Eclipse and Nova top level programs and flowcharts are given in Chapter 4 of Smith's thesis and in Appendix D of Russel's thesis.

Image Acquisition and Preprocessing

The equipment shown in Figure 2-3 was used to acquire and process images. The Octek 2000 video processing board connected to the Nova was used to acquire four-bit images from a black and white video camera. Once acquired, an image could be stored to disk, displayed on the monitor or printed on the

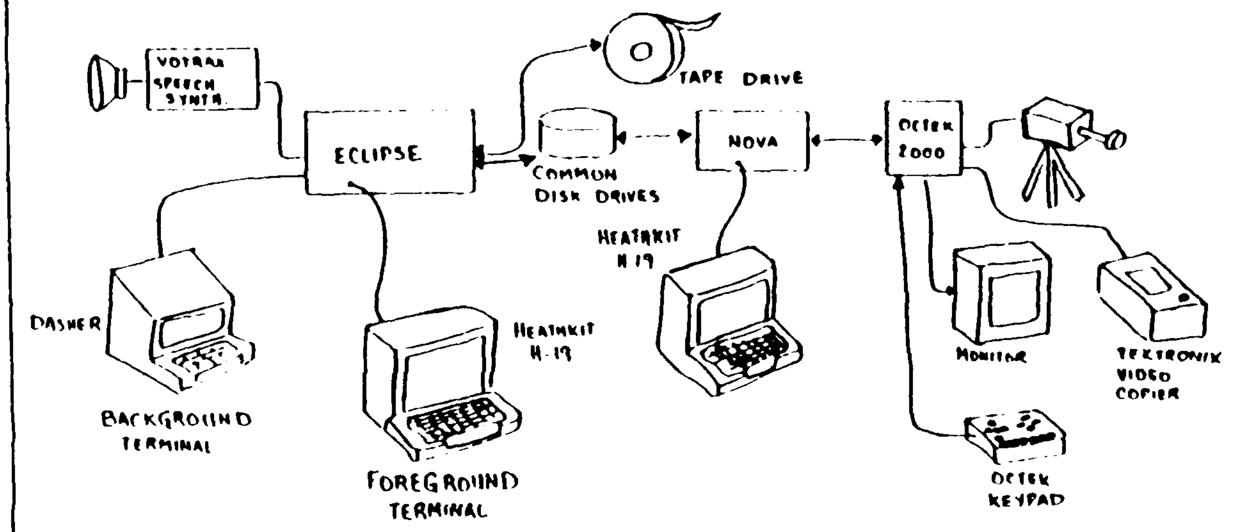


Figure 2-3. System Configuration (Russel, 1985:5-3)

video hard-copy unit. In Russel's thesis, image acquisition was accomplished with a fixed camera setup and layout. This layout is shown in Figure 2-4. The background was a plain piece of cardboard and the camera had to be calibrated to the brightness of this board (Russel, 1985:C-1). After taking a picture, the user provided the computer with the coordinates of the face by manually adjusting a box-shaped cursor around the subject's head as shown in figure 2-5 (Russel, 1985:B-9). In order to recognize the face, the computer had to divide the face into several separate windows. This windowing process is described in the following sections. Success in locating and windowing the face depended upon the contrast found in the scene and so the input scene had to be pre-processed to obtain a constant contrast value.

Preprocessing consisted of a contrast enhancement algorithm that sampled the pixel values in the center of the face and adjusted the contrast of the whole face based on the average of the center pixels (Russel 1985:5-7,4-22 to 4-27).

In Smith's thesis, images were acquired using the same equipment Russel used, but the setup shown in Figures 2-4 and 2-5 was not required. Instead of providing the computer with the face coordinates, the computer ran an automatic face location algorithm. The only requirement imposed on image acquisition was a camera calibration (Smith, 1986:B-6) and the background was allowed to vary.

Success in locating and windowing the subject's features still required a constant contrast value, so Russel's

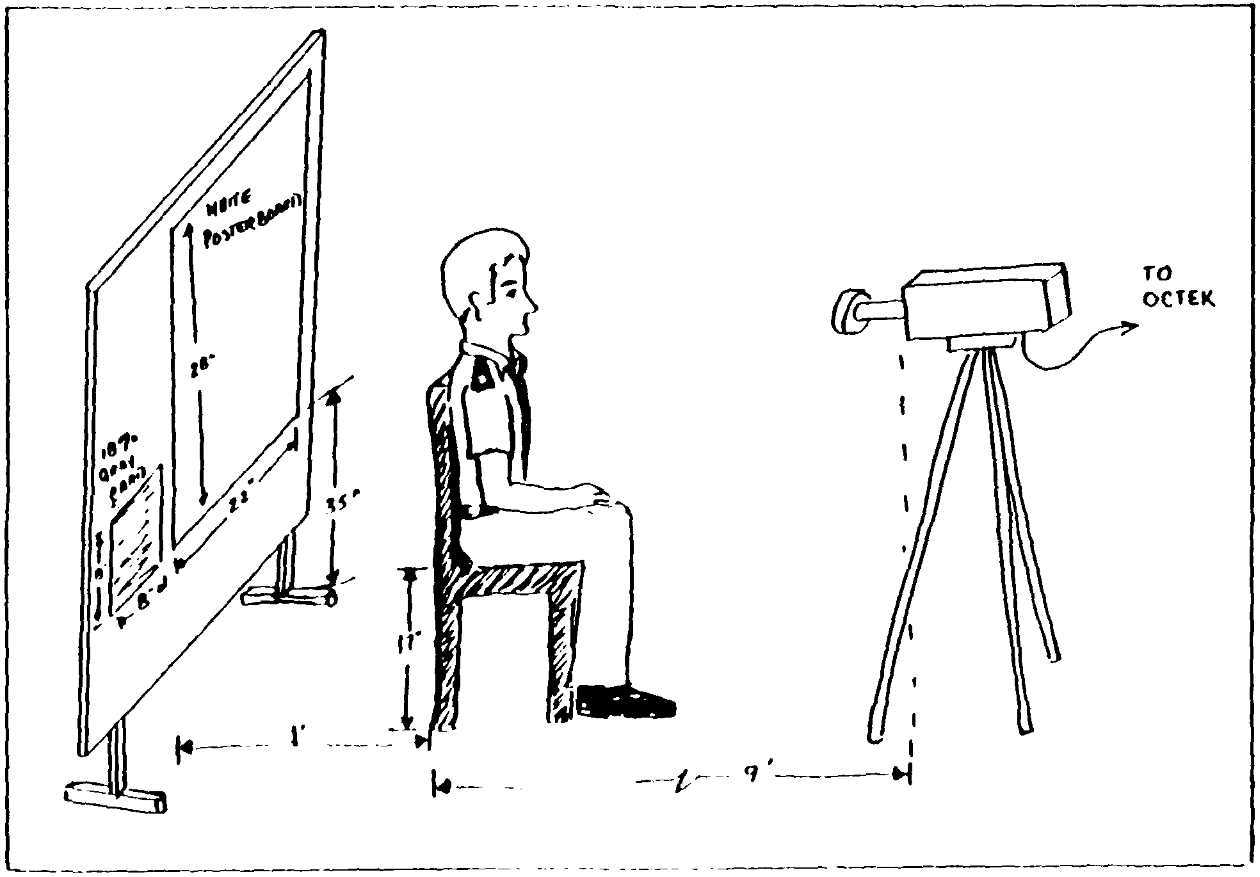


Figure 2-4. Studio Setup for Taking a Picture
(Russel, 1985:5-4)

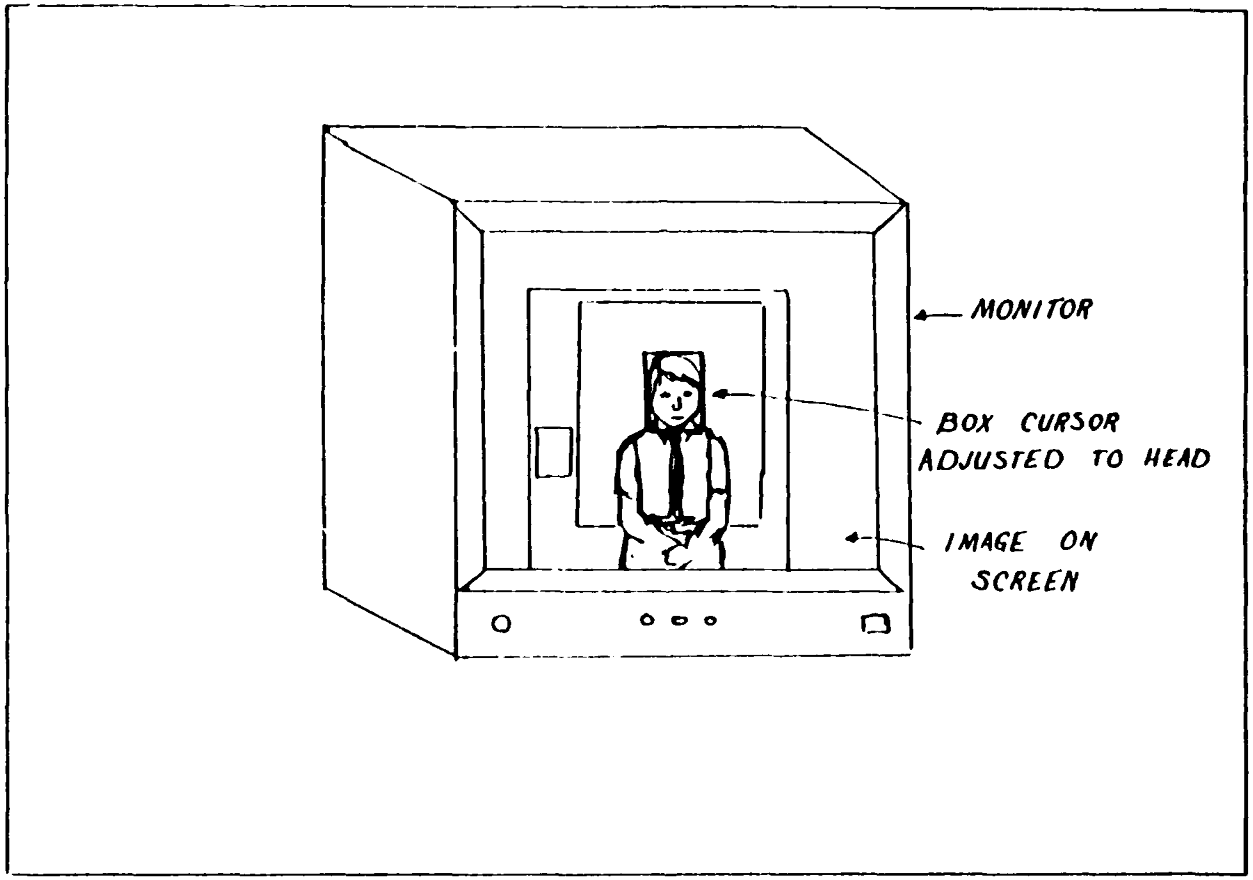


Figure 2-5. Cursor Adjustment on Image (Russel, 1985:5-5)

contrast enhancement algorithm was still used. The algorithm was applied to the scene after the face locator found most of the features of a face, in order to help it find the rest of the features (Smith, 1986:3-19). Then a slightly modified version of the contrast enhancement algorithm was applied to the face to improve the accuracy and repeatability of the windowing and recognition algorithms (Smith, 1986:4-15).

Face Location

There are two requirements of the face location algorithm used in the AFRM. The first is to ensure that only faces are passed to the recognition algorithm and that all other parts of the input scene are discarded. The second requirement is to find specific features on the face that need to be used by the windowing algorithm.

In Russel's FRM the first (face location) requirement was met by having the user position a block around the face as shown in Figure 2-5. The second (feature location) requirement was met by using an automatic feature location algorithm (Russel, 1985:5-40). The accuracy of the feature locations were dependent on the contrast of the input image, the set of rules within the location algorithm, and sometimes a manual correction entered by the user (Russel, 1985:B-23).

In Smith's AFRM the face location requirement was accomplished using an automatic "facefinder" algorithm. Feature location was accomplished as a part of the face location process. The facefinder works by searching an input image for certain facial characteristics called "signatures". The

facial signatures are present in most facial images and are rarely present when no face is present (Smith, 1986:3-1). Smith presented test results in Chapter 5 of his thesis that show how "face specific" the facefinder was.

The facial signatures are made up of the brightness variations in a scene that are consistently found when a face is present. The "eye signature" is made up of the three brightness maxima found around the eyes (one between and one to each side of the eyes) and the two brightness minima found in the center of the eyes. Figure 2-6 shows that these maxima and minima form a characteristic "W" shape when the brightness on a line through the eyes is plotted. Smith also defined a "nose/mouth signature" (Smith, 1986:3-14).

The development and calculation of the facial signatures was based in part on similar work (Bromley, 1977) in which specific features in mug file images were located using a signature technique. The signatures were generated by adding pixel values in each column of the image and plotting the results. Characteristic maxima and minima appeared at the center and edges of the face (Smith, 1986:2-6). Smith generated his facial signatures by extracting columns from an image and plotting the results of a one-dimensional gestalt calculation for each column (Smith, 1986:3-12).

After convolving the signatures with a gaussian function to smooth them, Smith applied a set of limits to determine if the signatures represented a face. The limits defined allowable variations in maxima and minima, the maximum

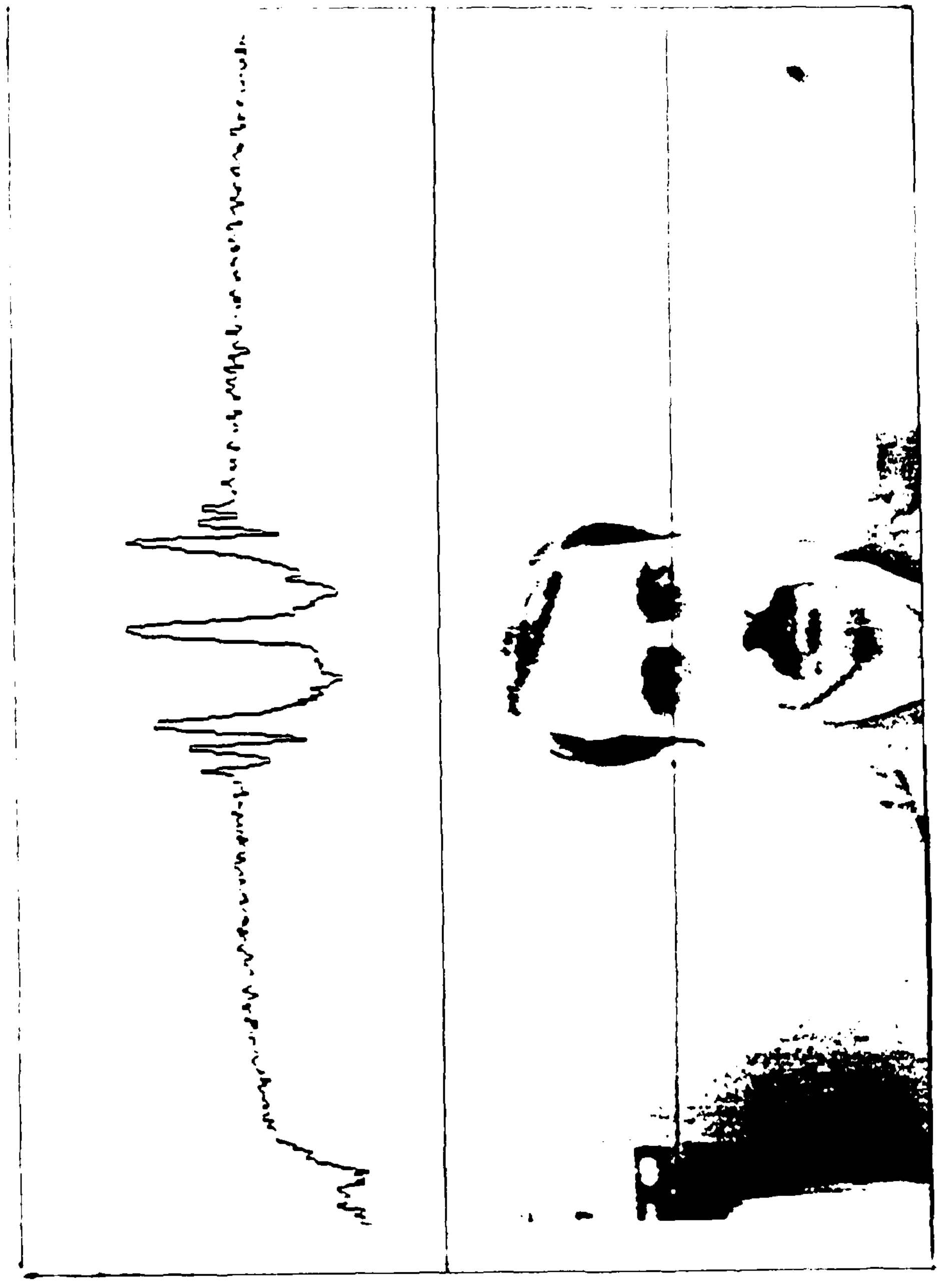


Figure 2-6. Eye Signature

distance ratios between various points on the signature, and the maximum variation of the slopes between maxima and minima (Smith, 1986:3-14,4-12).

Windows

Once the features were located, the face could be divided into windows. "Windowing" the face, or looking at small pieces, was required to separate similar faces and because the gestalt calculation had trouble with symmetrical faces (Russel, 1985:4-15 to 4-19). Russel used the following:

1. Left Half of Head.
2. Right Half of Head.
3. Right Side, Top of Eyes to Chin.
4. Right Side, Top of Eyes to Mouth.
5. Right Side, Top of Nose to Chin.
6. Right Side, Top of Head to Bottom of Eyes.

These windows are shown in Figure 2-7. Russel selected these windows based on the following research.

1. Russel's experiments with the gestalt calculation on whole faces showed that it could not distinguish a wide symmetrical face from a narrow face. The gestalt calculation is basically a center-of-mass calculation, so for symmetrical faces the center of mass always falls on a line drawn vertically through the center of the face. By gestalting the two halves of the face separately (windows 1 and 2), a change in the aspect ratio of the face will cause a change in the gestalt value (Russel, 1985:4-16).

2. Russel's literature review discusses the following experiments on human face recognition capability. Figure 2-8 shows a study of the ability of humans to recognize a face when shown only a part of the face (Goldstein and Makenberg, 1966). Some parts of the face yeilded higher recognition scores than others. Another experiment measured the number of times a baby looked at different features on its mother's face (Haith and others, 1977). Some features were used much more frequently than others.

Figure 2-7. Russel's Window Set (Russel, 1985:B-32)

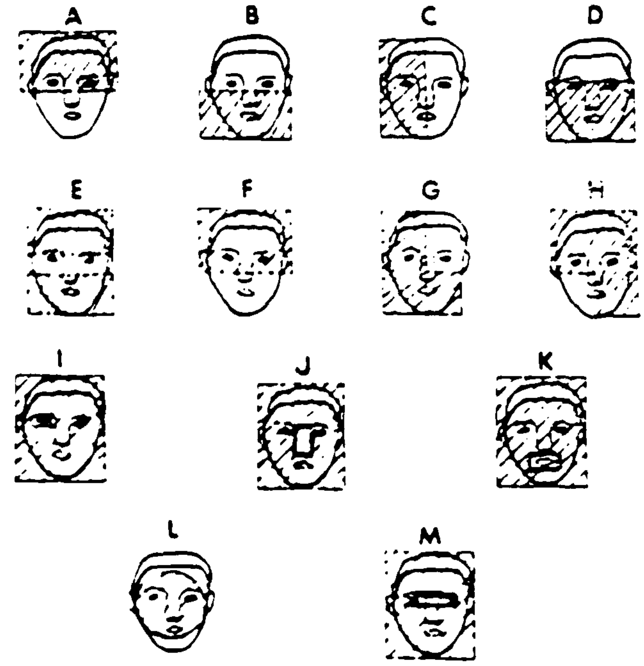


Fig. 1. Schematic representation of experimental conditions. An opaque mask was used to occlude crosshatched sections. Symmetrical features, such as Cond. H, were randomly varied to sample left and right sides.

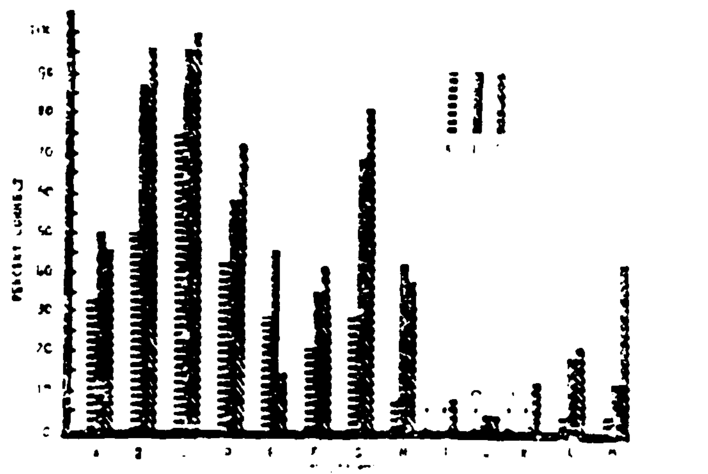


Fig. 2. Per cent correct identifications by kindergarten (K), first (1) and fifth (5) grade subjects on condition A to M.

Figure 2-8. Recognition of Person From Parts of Face (Russel, 1985:2-19)

Smith used a different set of windows because he had less feature information available. The facefinder was based on signatures that located the internal features only (eyes, nose, and mouth). Location of the edges of the head could not be obtained because the background was no longer a constant value. Figure 2-9 shows the set of windows selected by Smith.

Gestalt Calculation

The gestalt transformation is the heart of the AFRM. The results of this calculation provide the data needed to recognize the faces in the input scenes. Chapter 4 of Russel's thesis discusses the original gestalt transform (Routh 1985), and how it was modified for use in the FRM. The gestalt transform is basically a center-of-mass calculation where mass is represented by scene pixel values. The darker the pixel is, the more mass it has and therefore dark pixels will have more influence than light pixels on the location of the center-of-mass (a negative of the image is used so that the dark pixels become the larger mass values).

Figure 2-10 shows how a one-dimensional (1-D) gestalt transform was implemented. A point-by-point multiply and add (dot-product) was calculated between a gaussian function and an input waveform (2-10 b,c). The result was one element in the output array (2-10 d). By shifting the gaussian to the left and taking the dot-product again, the next value in the output array was calculated. Figure 2-10 shows the gaussian

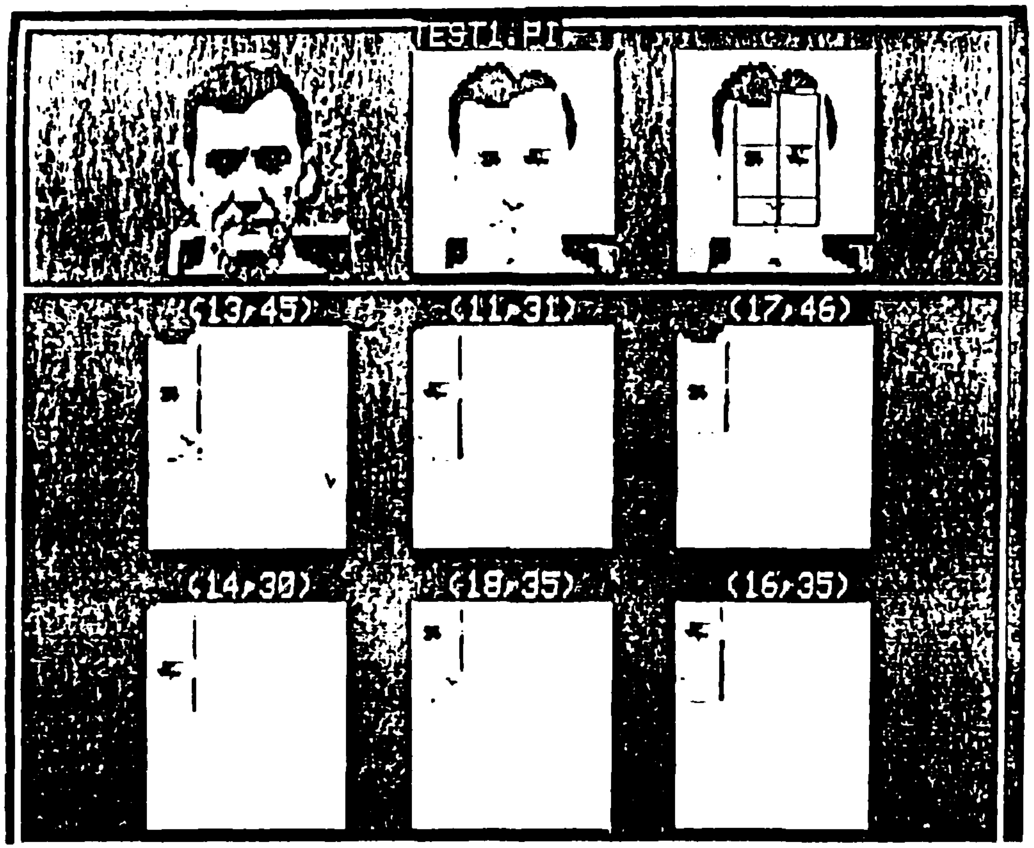


Figure 2-9. Smith's Window Set (Smith, 1986:3-32)

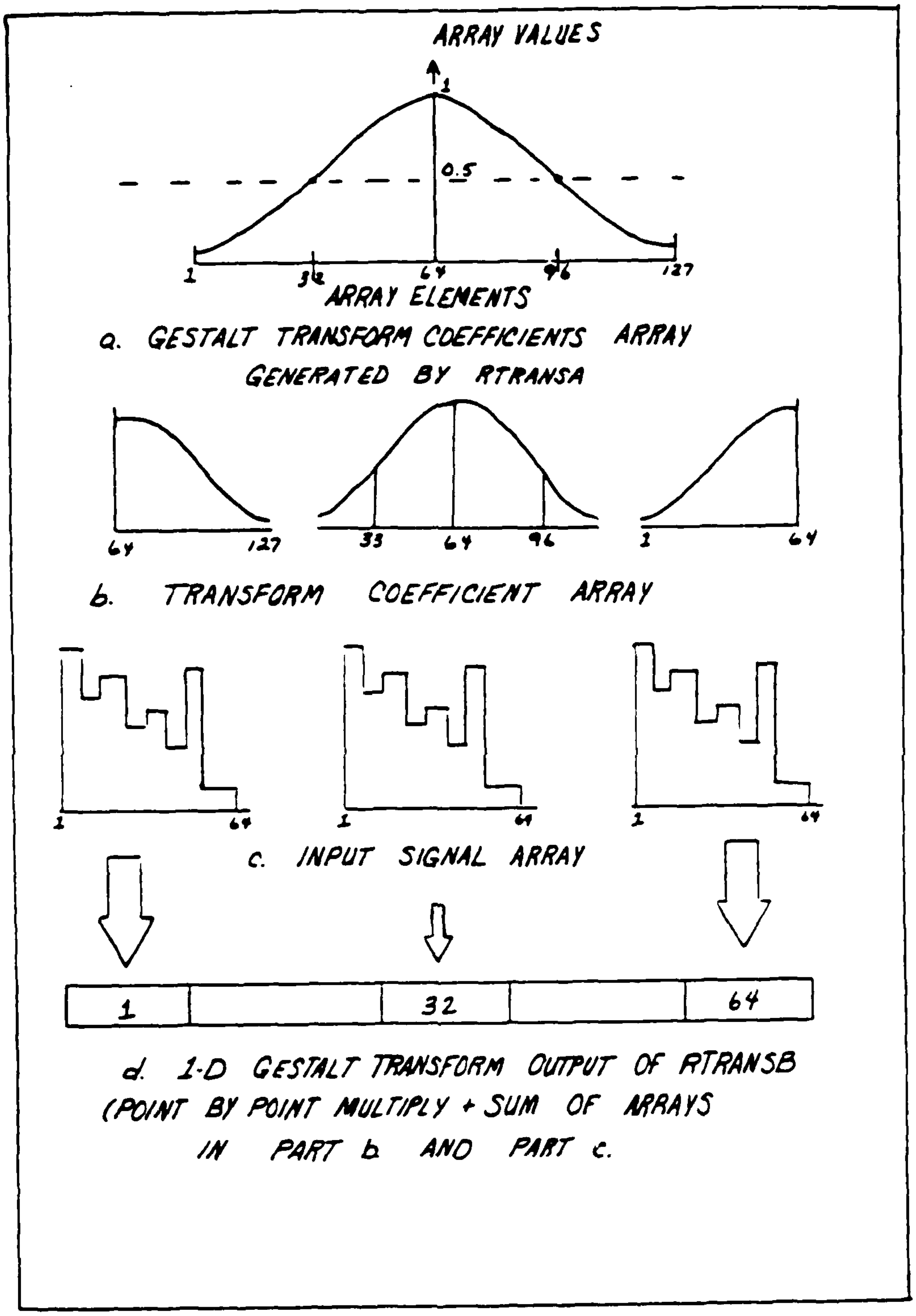


Figure 2-10. 1-D Gestalt Transformation Process
(Russel, 1985:5-43)

function used for calculating three elements (1, 32, and 64) of the output array. To calculate the gestalt for a 2-D image, the 1-D transform of each row of the image was calculated. The resulting arrays became the new image and the 1-D transform of each column was calculated. The gestalt value was the location of the maximum value in the resulting array as shown in Figure 2-11 (Russel 1985:5-42 to 5-46).

A final operation was performed to scale gestalt values to make the FRM size-invariant (size of face was allowed to vary). Figure 2-12 shows a window that was placed into a 64X64 array and gestalted. The scale factor (SF) applied to the gestalt value was the maximum scale factor that allowed the window to remain in the 64X64 array (Russel, 1985:4-10). The final gestalt value was calculated as follows:

$$SF = 64/A \quad \text{where} \quad A = \max(X\ WINDOW,\ Y\ WINDOW)$$

$$\text{Final Gestalt } (X\ ,\ Y) = (X' * SF\ ,\ Y' * SF)$$

Recognition

In order to identify and pull one face out of a group of faces, the AFRM has to be trained with the whole group. This was accomplished by setting up a database of gestalt values for a group of people. The database was loaded, "trained", with the gestalt values from 4 images of each individual. When a new face is entered into the AFRM and gestalted, the gestalt values are compared to those in the database. The name assigned to the new gestalt values is the name belonging to the closest set of gestalt values found in the database.

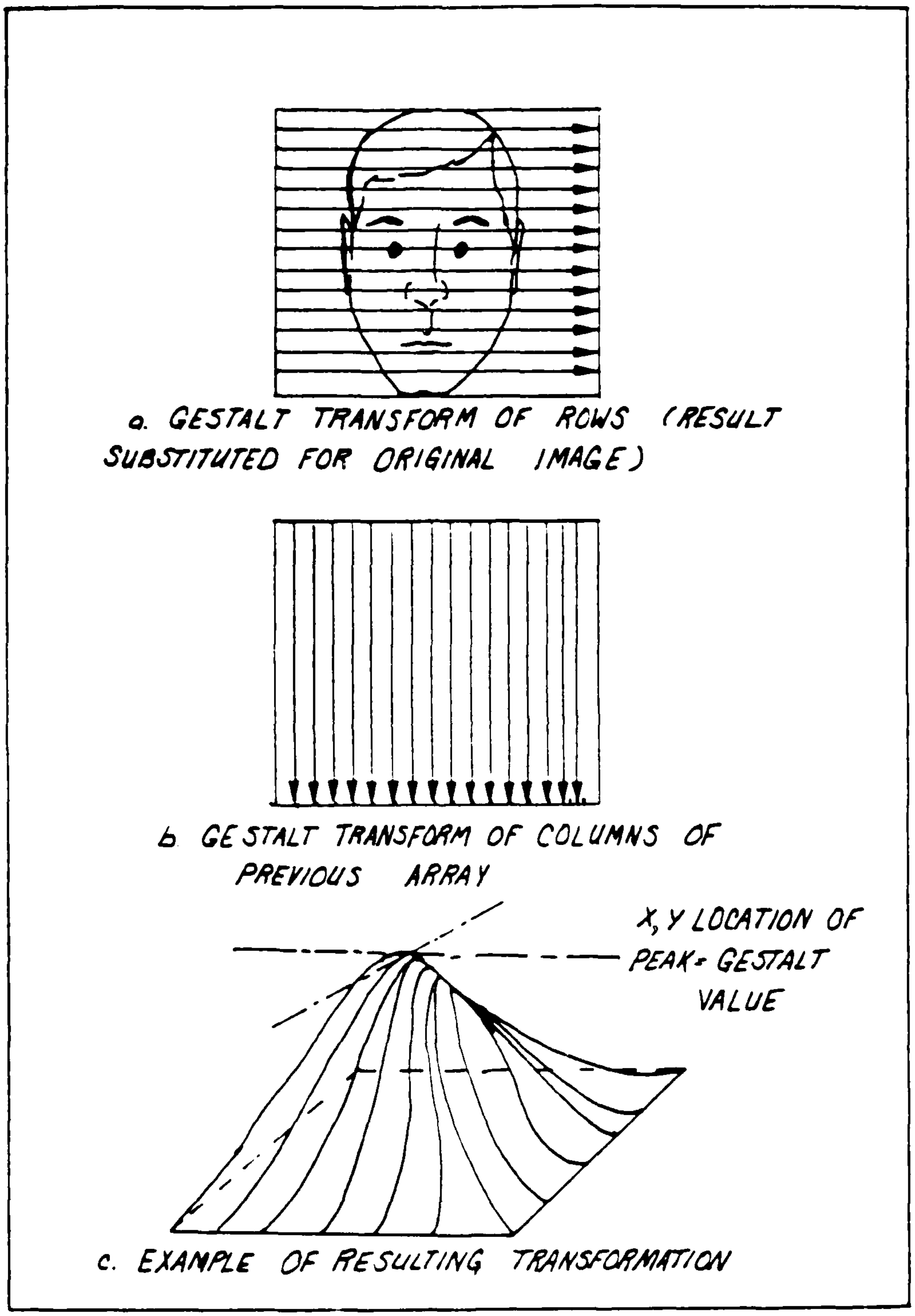


Figure 2-11. 2-D Gestalt Transformation Process
(Russel, 1985:5-45)

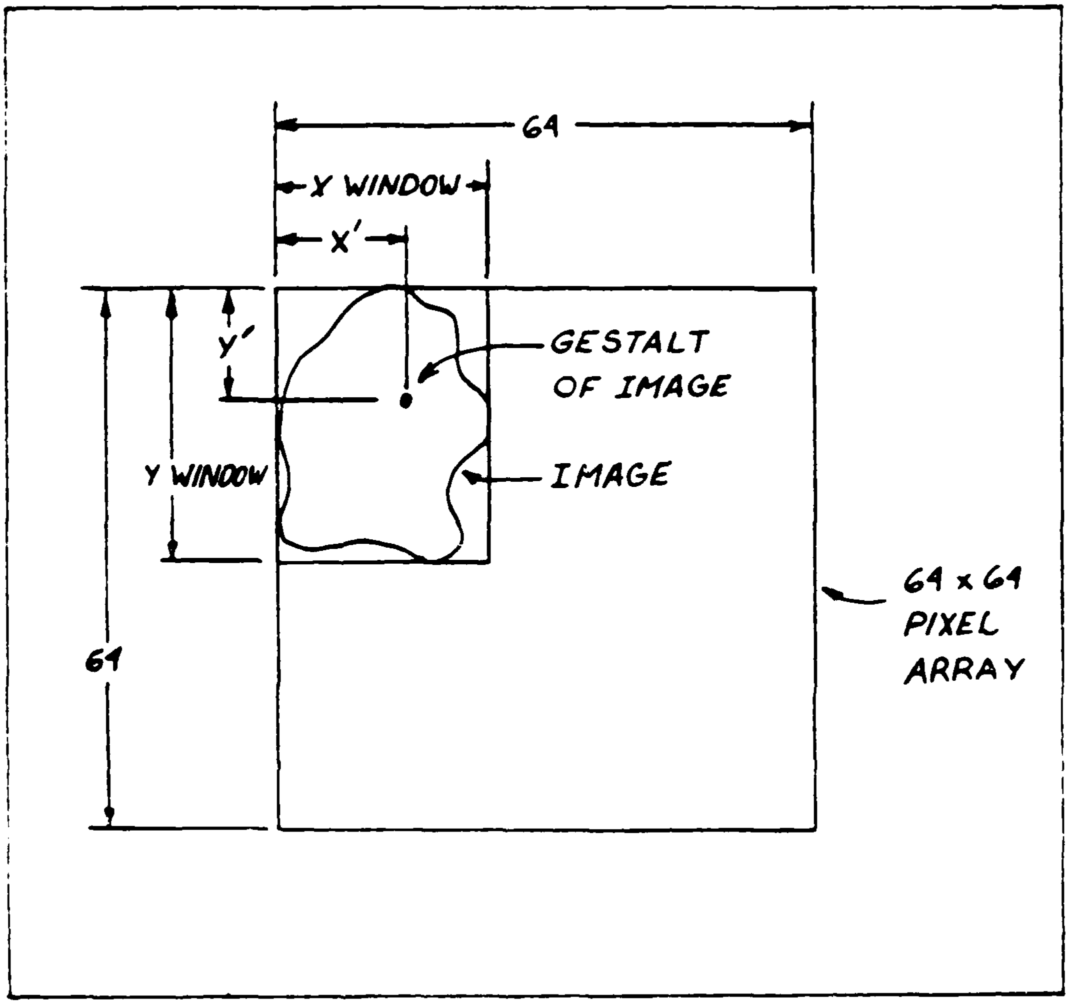


Figure 2-12. Dimensions Used for Scale Invarience Calculation (Russel, 1985:4-11)

If this name is correct then the AFRM has recognized the individual. Sometimes the first choice is not correct, but the AFRM is usually close. By rank ordering the individuals in the database from closest-to-the-input to farthest-from-the-input, a measurement of the "goodness" (Russel, 1985:2-3) of the system can be made. This measurement, called the Average Reduction in Uncertainty (Russel, 1985:6-8), tells how close the AFRM came to identifying the face correctly. Table 2-1 shows Russel's recognition results. With a database of 20 individuals, a 99% reduction in uncertainty was obtained.

Table 2-1. Russel's Test Results (Russel, 1985:6-9)

Number in Database:	20
Number Recognized as 1st Choice:	18
Number Recognized as 2nd Choice:	1
Number Recognized as 3rd Choice:	1
Absolute Correctness =	0.90
Average Reduction in Uncertainty =	0.9925

In order to achieve "human-like" recall capabilities, the database was setup so that gestalt values would directly provide the name of the individual in the input image. The structure of the database is shown in Figure 2-13. Since the training for each individual was done with multiple images (up to 4) there is an area on the surface of the database structure where each individual could be mapped. Instead of mapping the individual into more than one coordinate location the AFRM trains only one location with the individual's data.

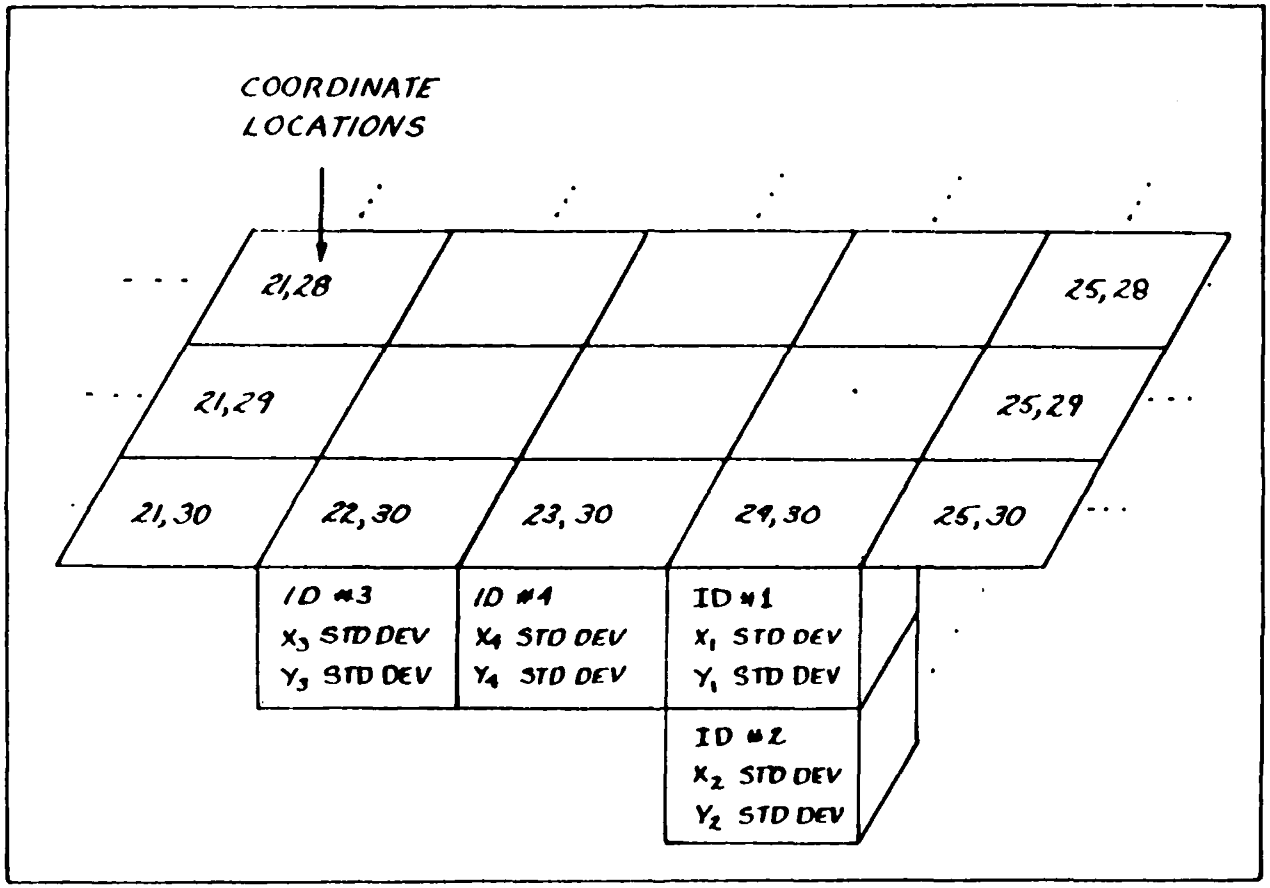


Figure 2-13. Example of Data Storage in a Recognition Database (Russel, 1985:4-31)

The data includes the size of the area that this individual's gestalt values are spread over (as X,Y standard deviations), and the individual's ID number. Now by combining the six window gestalts into one coordinate location and searching for ID numbers within a specific range around this location, an ordered list of names is generated. The recognition time is fixed by fixing the size of the searching range. Russel's thesis gives a complete description of the design and implementation of the database.

Summary

This chapter presented a review of past face recognition efforts at AFIT. This review should give the reader enough background on the AFIT face recognition system to understand the evaluation and enhancements presented in Chapter 3.

III. Evaluation and Enhancement

This chapter evaluates the AFRM discussed in Chapter 2 and presents enhancements made for speed and accuracy. The sections in this chapter have the same titles and order used in Chapter 2.

System Environment

Re-hosting the AFRM onto a new computer system was an important part of this thesis effort. This was required for several reasons. First, the Data General system shown in Figure 2-3 has outlived its usefulness to AFIT and may soon be removed from the Signal Processing Lab (Kabrisky, 1987). Second, the memory limitations and communication problems within that system resulted in slow-running programs and complicated programming techniques. Third, a new environment was needed to allow some of the enhancements added as part of this thesis effort.

A Micro-VAX II computer in the Signal Processing Lab met all the hardware requirements needed by the AFRM including:

Memory: A 9MByte main memory, Three 71 MByte hard disk units, and a TK50 tape drive.

Image Processing: An FG-100-Q Image Processing system (with software library), and an RGB monitor.

Software: MicroVMS 4.4 operating system, DECnet, VAX Fortran, LISP, and C.

Access to: A Video Hard Copy Unit, Printers, and an RS-232 connection to the Data General.

Choosing a software language was based on minimizing the effort of rewriting the AFRM (by using a similar language), and allowing easy interface to the hardware components. Of the three languages available on the MicroVAX (Fortran, C, and LISP), Fortran and C were chosen as the easiest possible replacements for the Data General Fortran IV and Fortran V. C was chosen over Fortran because the software library for the image processing board on the Micro-VAX is written in C and extensive use of this library is necessary.

Image Acquisition and Preprocessing

In order to recognize a subject, the AFRM must be given an image of the subject's face with no significant tilt or rotation of the subject's head. With no other constraints imposed on the image, the AFRM is required to locate and recognize the subject. In order to help the AFRM accomplish this task quickly, accurately, and consistantly, several preprocessing steps can be performed on the input image.

Moving Target Indicator.

In Russel's FRM, the user was required to identify the location of the face by positioning a box-shaped cursor on the video monitor. The only part of the image used by the FRM was the part inside the box. Smith provided the whole scene to the AFRM and added an automatic location algorithm to locate a randomly positioned face in a random background. This location algorithm could take anywhere from 5 to 30 minutes to find the face in the scene (Smith, 1986:B-9) and

could not separate the edge of the subject's head from the background.

As part of the thesis effort reported here, a moving target indicator (MTI) algorithm was added to the AFRM as the first step in processing the scene, with the following assumption:

> Faces may be present on moving targets but are never present when there is no motion. (The motion must occur between the acquisition of two consecutive scenes and the user will be allowed to bypass the MTI step in order to process previously stored or "still" photos).

Using this assumption, the time required to locate a face would be greatly reduced by searching only a portion of the scene. In addition, the MTI algorithm might enable the separation of the edges of the head from the background. Figures 3-1 through 3-5 show how the MTI works. Figure 3-1 shows two scenes, one with a subject and one without. The top of Figure 3-2 shows the result of a point-by-point subtraction of one scene from the other. At this point the presence of a moving target is determined. The location of the target is the location of any non-zero pixel values in the resulting scene (everything described so far can be accomplished in less than one second). In Figure 3-3 a block has been drawn around the non-zero values and all these values have been changed to 255. This shows that some areas of the moving target had pixel values equal to the values in the background (holes) and that video noise in the backgrounds did not allow the backgrounds to cancel out (leaving spots). In Figure 3-4 all the holes have been

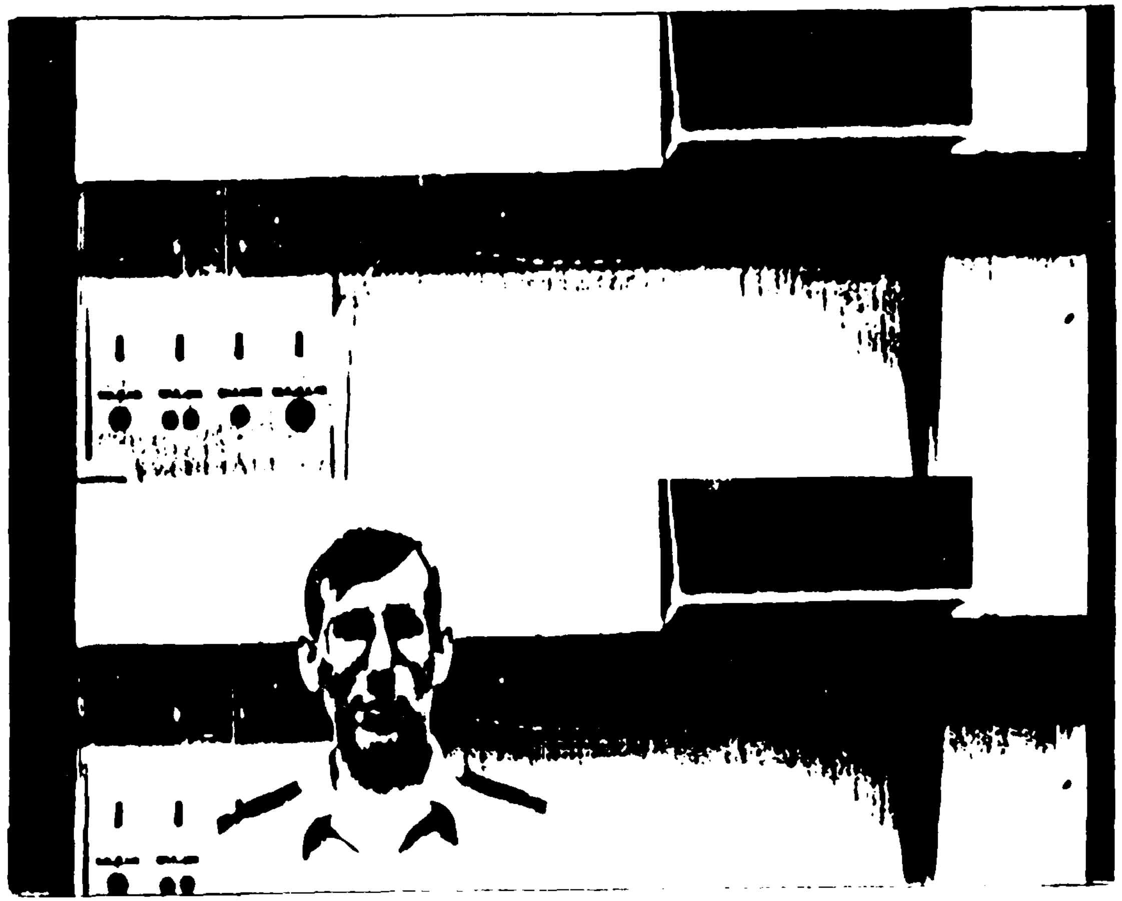


Figure 3-1. Input to MTI

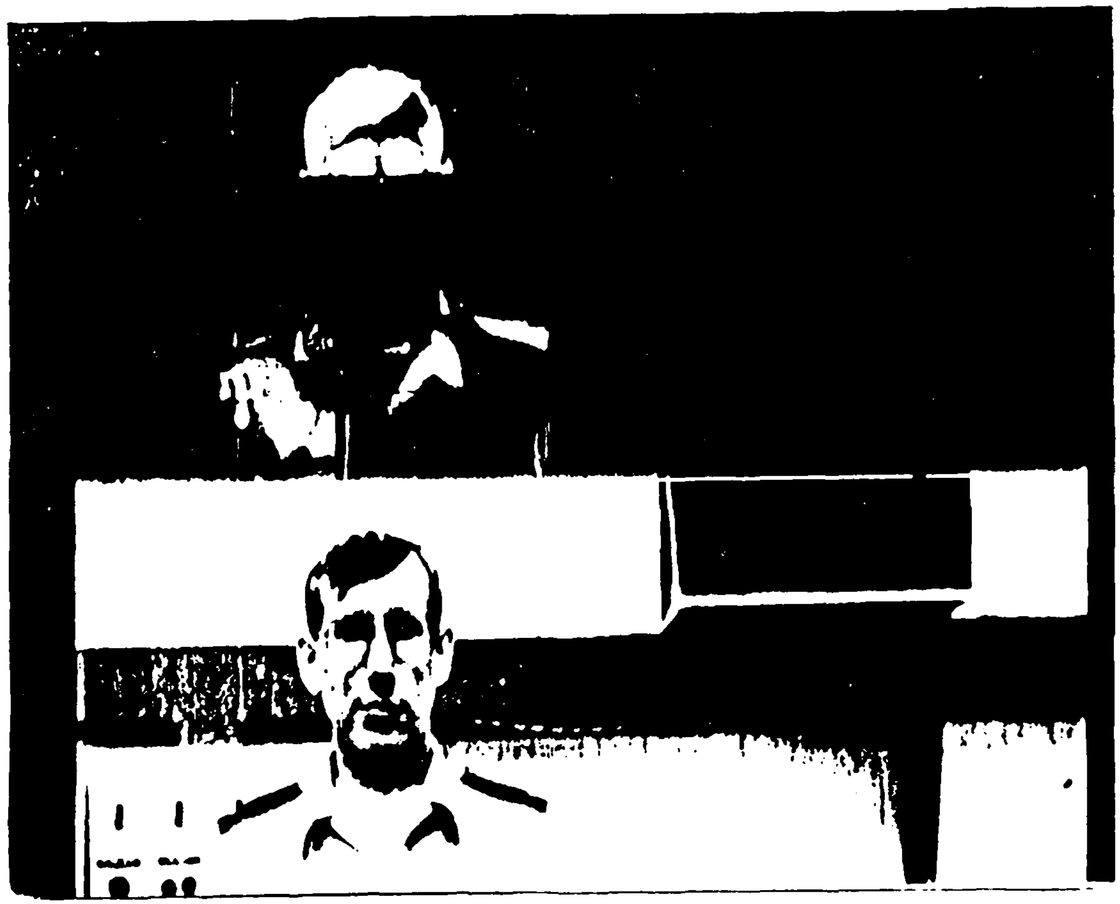


Figure 3-2. Scene Subtraction

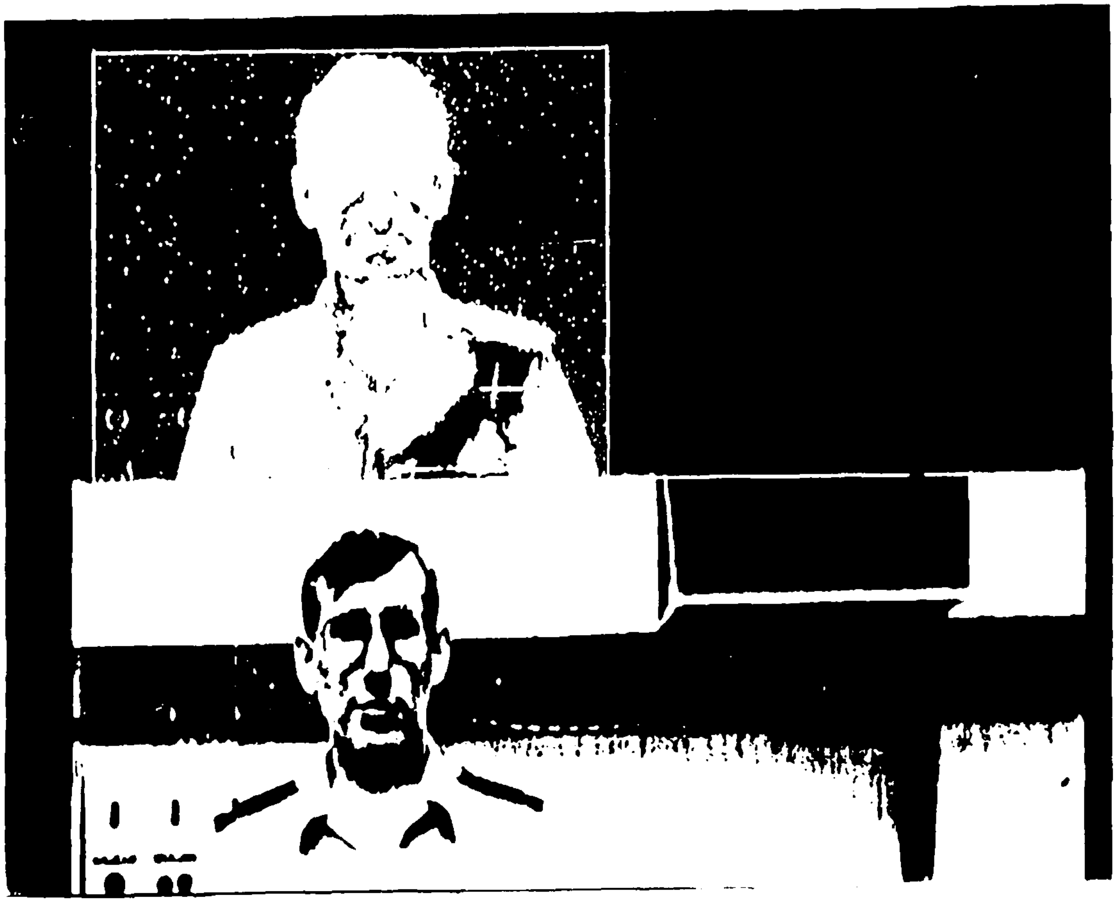


Figure 3-3. Non-Zero Pixels Set to 255

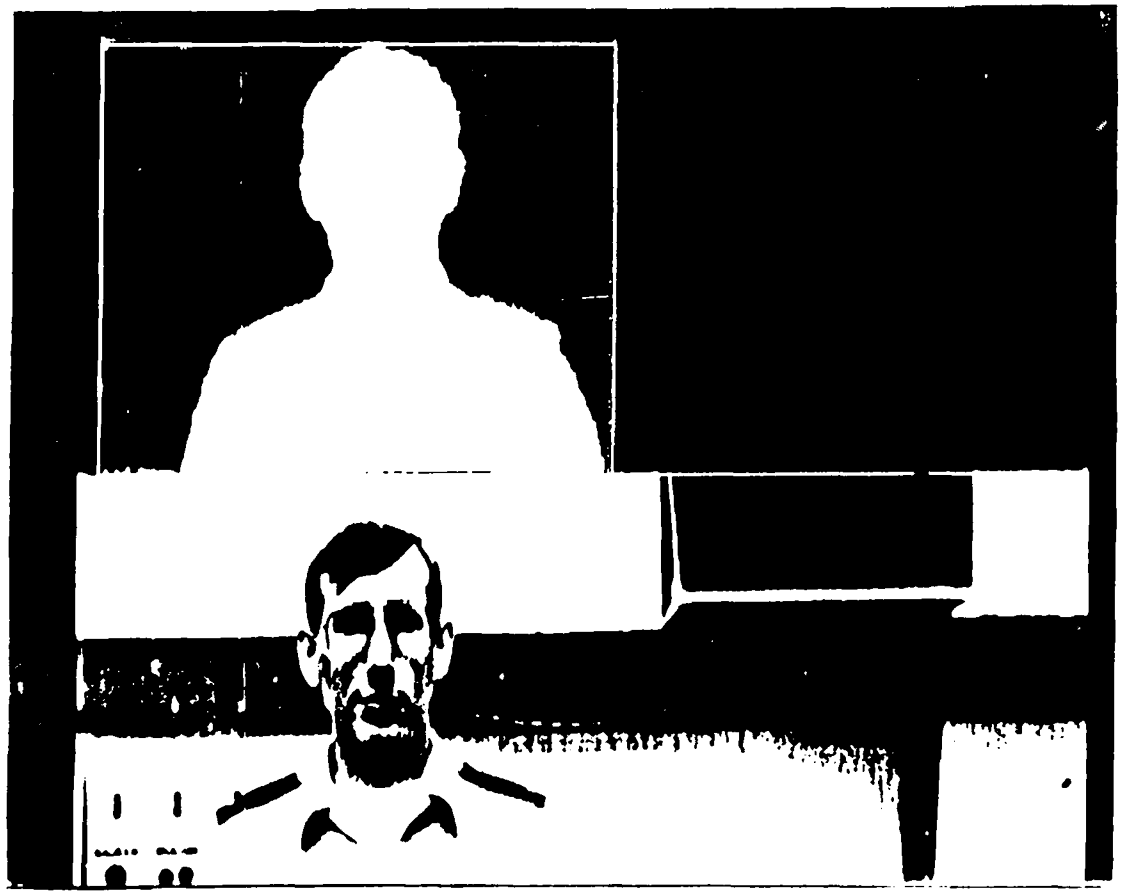


Figure 3-4. Target Mask

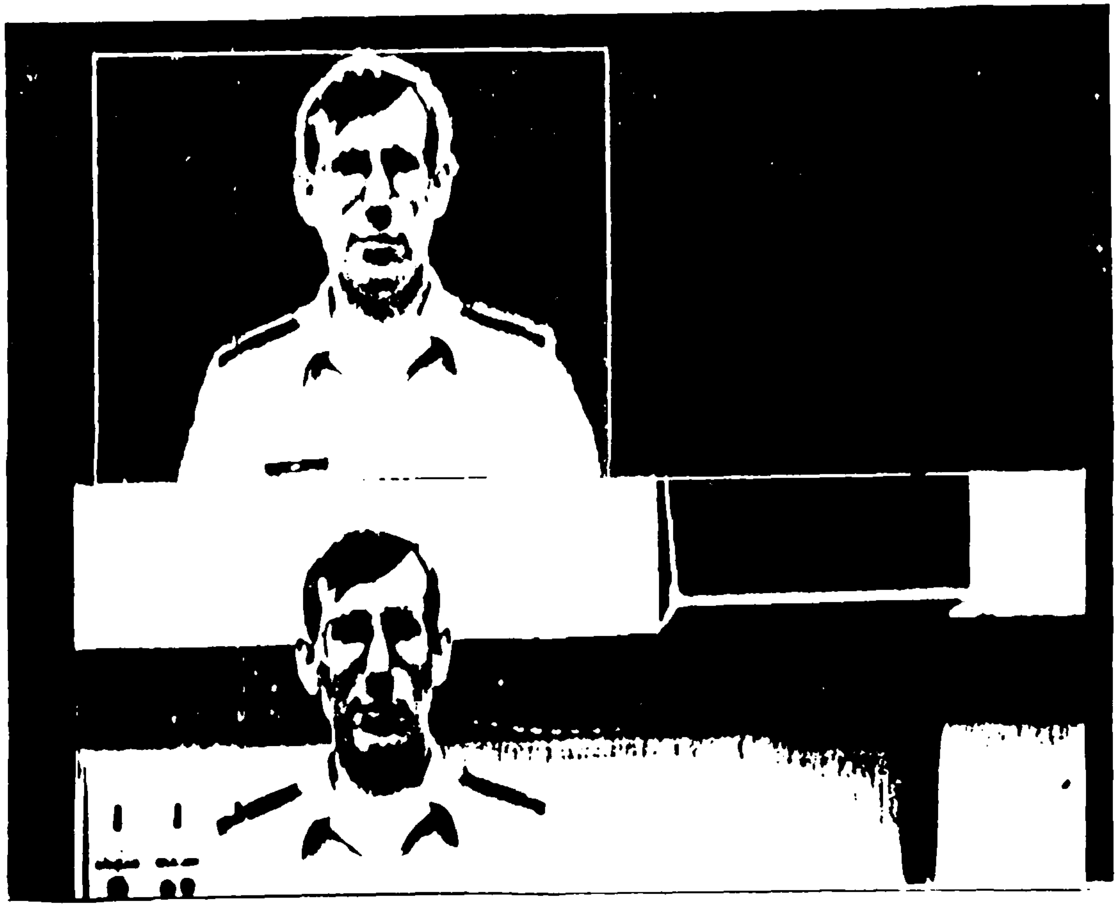


Figure 3-5. Target Separated From Background

filled and all spots eliminated to create a target mask. Figure 3-5 shows the results of a point-by-point logical AND performed between the bottom scene and this mask. This scene becomes the input into the AFRM face location algorithm. The results of testing the MTI are as follows:

1. The AFRM face location time has been reduced by a factor of M because it only searches inside the block.

   M = scene size / block size

2. The AFRM does no further processing on a scene if there is no moving target because the scene has been reduced to zeros (it can operate in a loop until a target is found).

3. Most edges of the head are closely (but not exactly) determined, but nothing is known about the bottom edge because the subject's body is part of the moving target.

4. All the holes in the mask (for this first example) were completely surrounded by a white area and so it was easy to determine what to fill in. Figure 3-6 shows a second example where a hole in the mask is at the edge. There was no way for the computer to decide whether this was a hole or actually the proper edge of the moving target (look near the hairline). In this case the random background matched a small region of the head causing its elimination from the input scene. This randomly occuring and undetectable event may result in an unrecognizable face.

Because of these results, in the present AFRM the MTI algorithm was implemented only to speed up the location of the face. The processing of Figures 3-3 and 3-4 had to be replaced by a more consistant method of determining the edges of the head, and the option of skipping the MTI altogether was provided to allow still photo processing. The software for the real-time subtraction is called

Figure 3-6. Example of a Hole in the Mask

SUB_DEMO.C, and for detecting and isolating of targets, is called MTI.C. This software is listed in Appendix B.

Elliptical Mask

The maximum information that can be provided to the AFRM for recognition of a subject's face is a frontal view of the subject's whole head. No information from beyond the edges of the head is allowed because this information comes from an uncontrolled background. The MTI algorithm can separate a moving target from the background, but it is not quite good enough to use with only brightness values from a black and white image. Smith chose to provide only the internal features of the head and this resulted in a lower recognition accuracy. The technique presented here, called an elliptical mask, is designed to provide the recognition algorithm with an approximation to a whole-head view of the subject (providing less information than the whole-head approach but more information than the internal feature approach). The goal is to give the face recognizer as much information about the subject as possible without adding uncontrollable background data. In order to create an elliptical mask, the following assumption is necessary.

> The head is elliptical (available software allows easy creation of only ellipses, circles or rectangles). The size of the head can be approximated if the size of the internal facial features are known.

Using this assumption and Smith's automatic face location algorithm (to find and measure the internal features), the size of the subject's head is approximated. An ellipse of

this size is drawn around the face and everything outside the ellipse is cleared to a constant brightness value. This new image is fed into the recognition algorithm just as Russel did with a whole-head on a pristine background. Testing the elliptical mask algorithm yielded the following result.

> The ratio between facial feature size and head size is not a constant. Figure 3-7 illustrates this with three subjects having the same size heads. This figure also shows that features may not be found in a constant location on the head.

Because of this result, the ellipse size and center location had to be adjusted so that it would not extend beyond the edge of any subject's head. The proportions and placement of features for a typical head were obtained from a drawing course instruction book (Edwards, 1979:143-145). Figure 3-8 shows the amount of each head made available by the final version of the ellipse algorithm. It can be seen that different amounts of data are available to the recognition algorithm for each subject. This is good because the internal features (faces) of all the subjects are identical (the face of subject #3 is a scaled up version of the others and the face recognizer is designed to be scale invariant). The face recognition algorithm cannot "see" the hair of subject #1 in this image, but this will be consistant for all photos of subject #1 and so this is not a problem. Being able to "see" data outside of the internal feature area, (more data for some faces than others) has allowed the AFRM to distinguish between three subjects that could not be separated using only internal feature information.

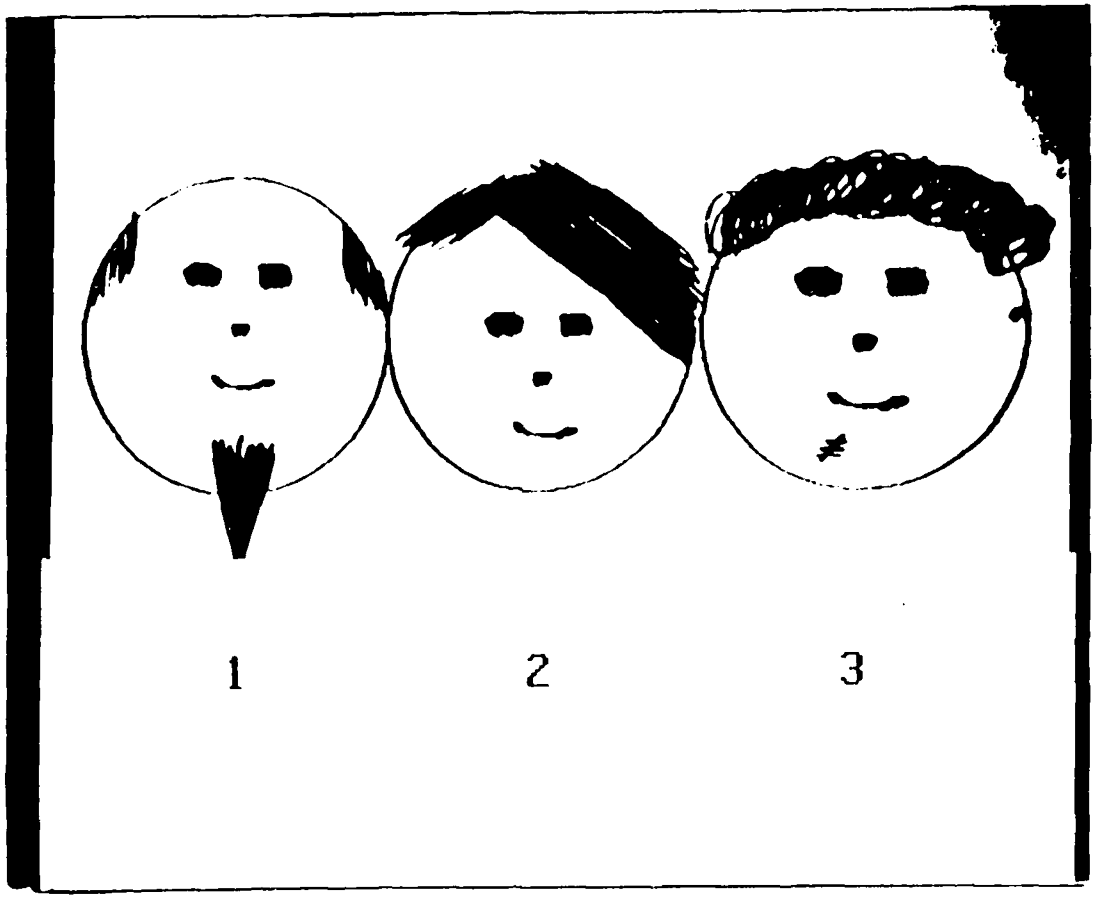


Figure 3-7. Three Faces With Identical Internal Features

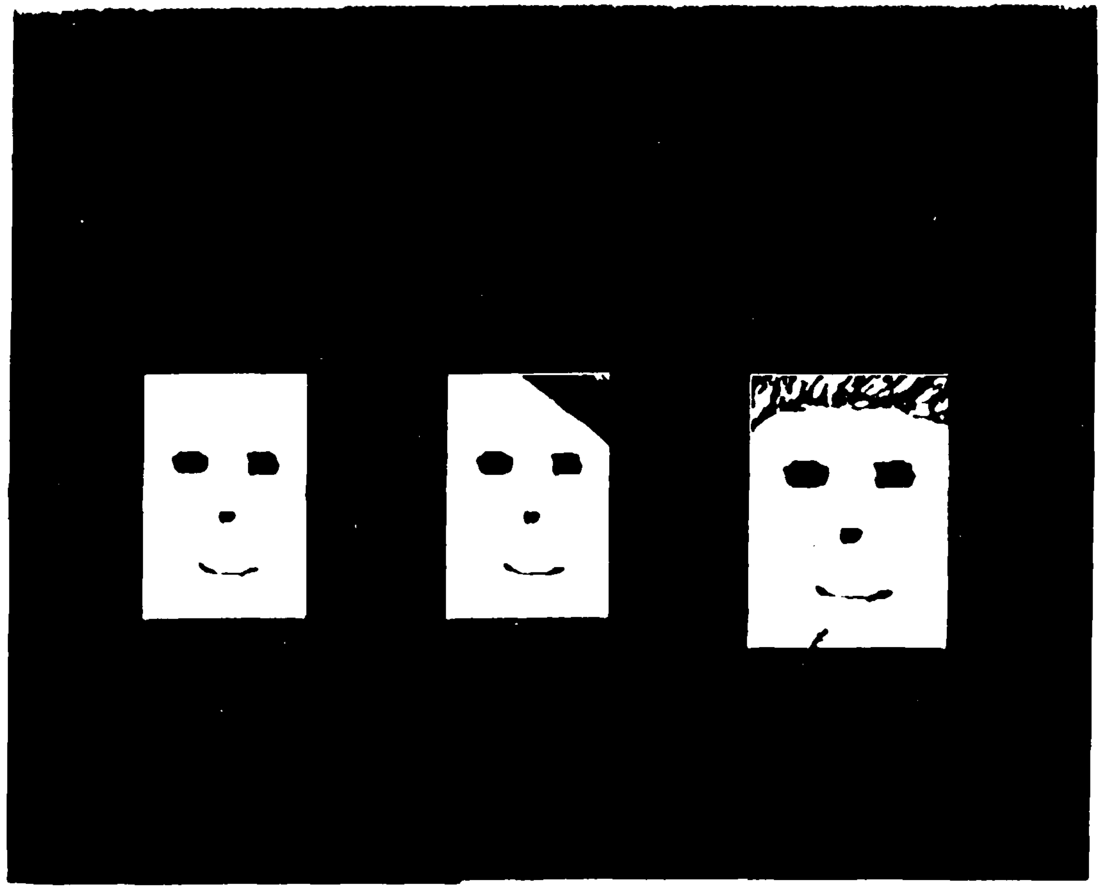


Figure 3-8. Data Available to Recognizer

The software used to calculate the size and location of the ellipse is included as part of the subroutine FACEMAP in the code for the AFRM, called FACE.C in Appendix B.

Brightness Normalization

When Smith's face location algorithm was re-coded on the Micro-VAX, a modification was made to make it invariant to the overall brightness of the input scenes. This allowed the location algorithm to locate a face in both dark and bright settings as shown in figure 3-9. This was done by comparing the eye signature minima and maxima to the value of the first maximum found (all others had to be within a specific range of the first). The value of the first maximum is the local brightness of the signature, and the eyes are "dark" compared to this value.

Further testing showed that the face location algorithm was prone to false alarms when presented with a certain class of input objects (which is discussed in the next section). The solution to this false alarm problem was to change the face locator from a one-dimensional (1-D) signature analysis algorithm, into a 2-D object analysis algorithm. This required that an eye be dark compared to brightness values all the way around the eye, no longer at just one point. The new face location algorithm had to find dark objects in a 2-D scene rather than dark points on a 1-D signature. A "dark" object is defined as an object that has a lower brightness value than the values of all objects surrounding it. For example; a brightness value of 20 is not considered dark if

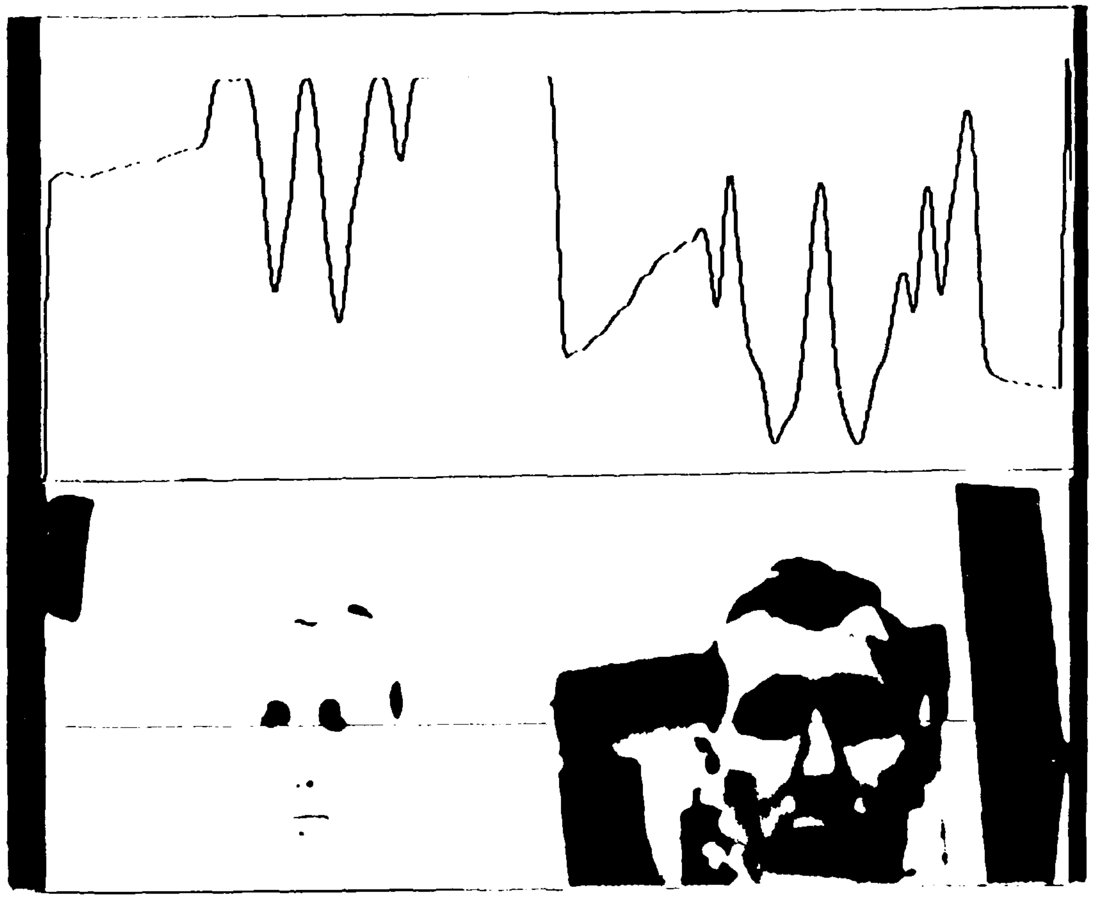


Figure 3-9. Signatures at Different Brightness Levels

all the values around it are 10, but a value of 100 is dark if it is surrounded by values of 200.

The brightness normalization algorithm calculates the average brightness in a square-shaped neighborhood around a pixel and resets the pixel value as follows:

$$\text{pixel value} = 128 + (\text{pixel value} - \text{neighborhood average})$$

In this way the "darkness" of the pixel is measured relative to a fixed average value (128). This is done for every pixel in the scene (each having its own neighborhood). Figure 3-10 shows an input scene with 4 different regions of brightness. Normalizing this whole scene results in Figure 3-11. This figure shows that the recognizability of a face is not dependent on the overall brightness of the face, and it will be scenes like Figure 3-11 that are input into the face recognition algorithm. The face location algorithm is only looking for dark objects, so a second step in brightness normalization is to decide whether a pixel is dark or light relative to its surroundings. From Figure 3-11 it is easy to see that anything below the fixed average of 128 is dark and anything above is light. By setting a threshold value just below 128 and comparing all pixels to this value, a binary (light/dark) scene like Figure 3-12 is easy to generate. This scene shows that all facial features were dark relative to their surroundings (this is why the signature technique worked).

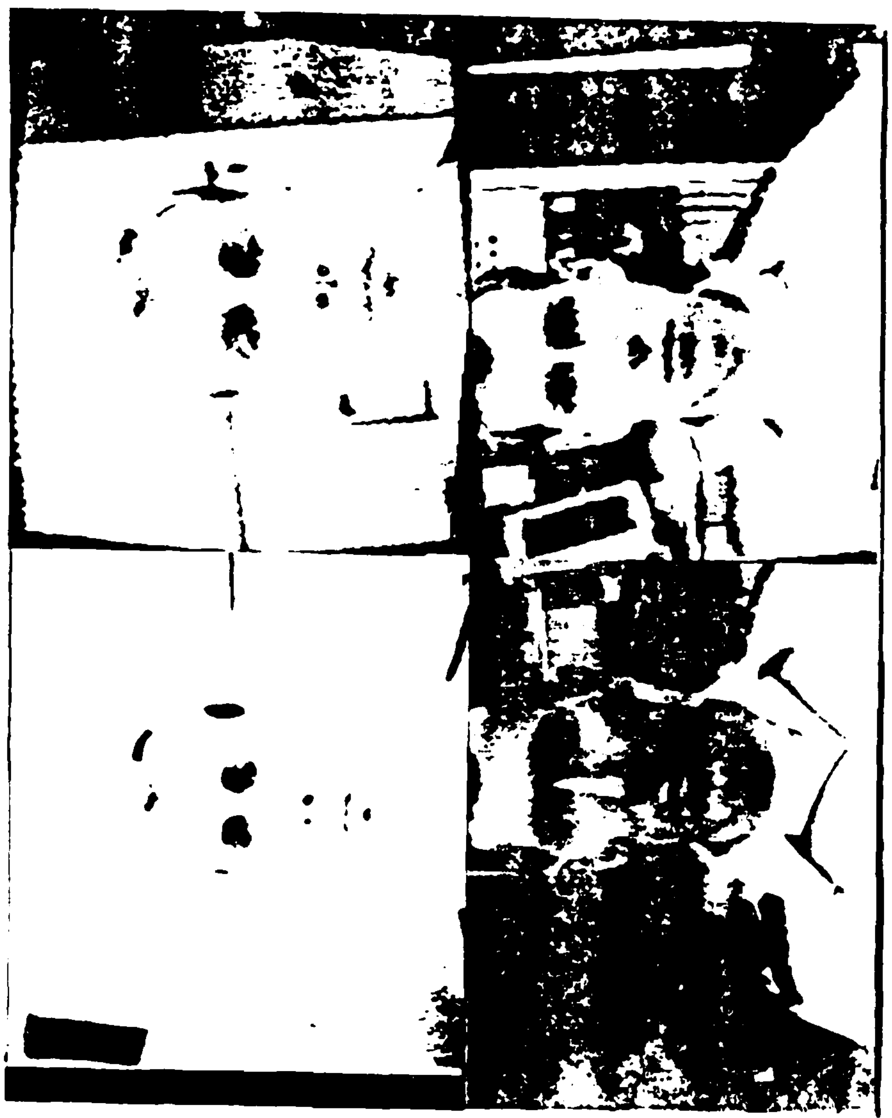


Figure 3-10. Input to Brightness Normalization

Figure 3-11. Output From Brightness Normalization

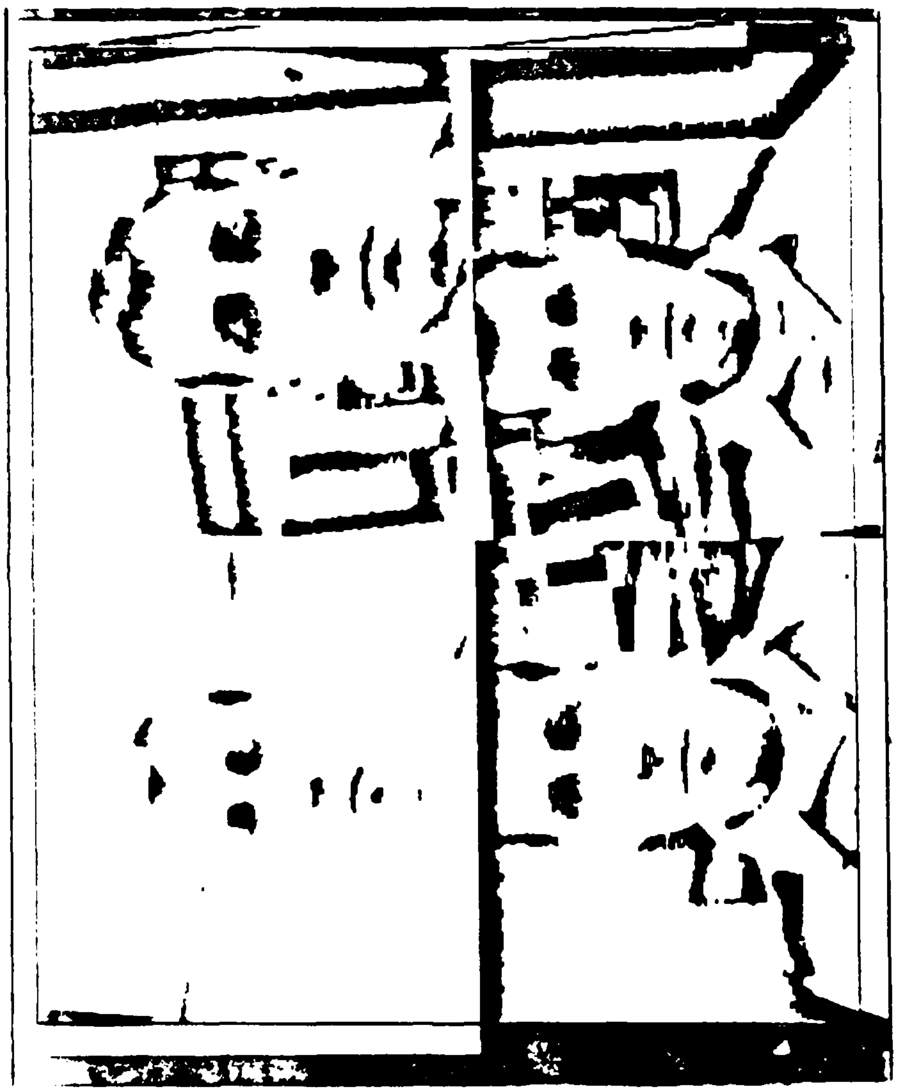


Figure 3-12. Binary (Light/Dark) Scene

The first half of the brightness normalization algorithm is a reasonable pre-processing step for a CTT-based and "human-like" face recognizer. The algorithm fits a human model if it is thought of as one of the pre-processing steps performed by the retina (Werblin, 1973:71-79) before sending the image to the brain. Asking a subject to point to dark objects in Figure 3-11 verifies the human ability to make the light/dark decision (the second half of the algorithm) but no assumptions are made about where this decision occurs in the human.

The code for this algorithm, without the binary decision part, is listed in Appendix B as BRIGHT.C. Appendix E gives a complete analysis of this algorithm. The whole algorithm is included as a subroutine in FACE.C called BRIGHT_NORM.

Contrast Enhancement

In Smith's AFRM, contrast enhancement was used to help the face location algorithm. It was also used by both Smith and Russel to more accurately locate the edges of features prior to windowing. After calculating the window locations, both authors could have taken the data for each window from the original (non-enhanced) scene. However, the contrast enhanced faces presented much more consistant data to the recognition algorithm because slight shadows and reflections were removed from the face. The new AFRM presented here no longer needs to enhance contrast in order to locate and window faces, however it is still used prior to recognition.

The recognition algorithm takes faces that have been

brightness normalized and contrast enhances them using an ITEX library function called HISTEQ. This function generates a histogram of a specified sample ırea on an image and then uses this histogram to modify the brightness values of the entire image (ITEX-100, 1986:9-27). In this case the sample area is the internal feature area starting just below the eyes and ending at the center of the mouth. The result of contrast enhancement is shown in Figure 3-14. This result is close to the enhanced faces in Figures 2-7 and 2-9.

Smoothing

Before Smith's face location algorithm could evaluate a signature to see if it represented a face, Smith found it necessary to convolve the signature with a gaussian function to smooth it. This was to eliminate noise that added extra minima and maxima to the signatures as shown in Figure 2-6. Figure 3-9 shows a signature after smoothing. On the Micro-VAX, the ITEX library function "BLUR" was used for smoothing the image in this work. This function convolves the image with the following kernel.

1	1	1	1	1
1	2	2	2	1
1	2	1	2	1
1	2	2	2	1
1	1	1	1	1

Figure 3-13. Pill-Box Kernel for BLUR (ITEX-100, 1986:9-25)

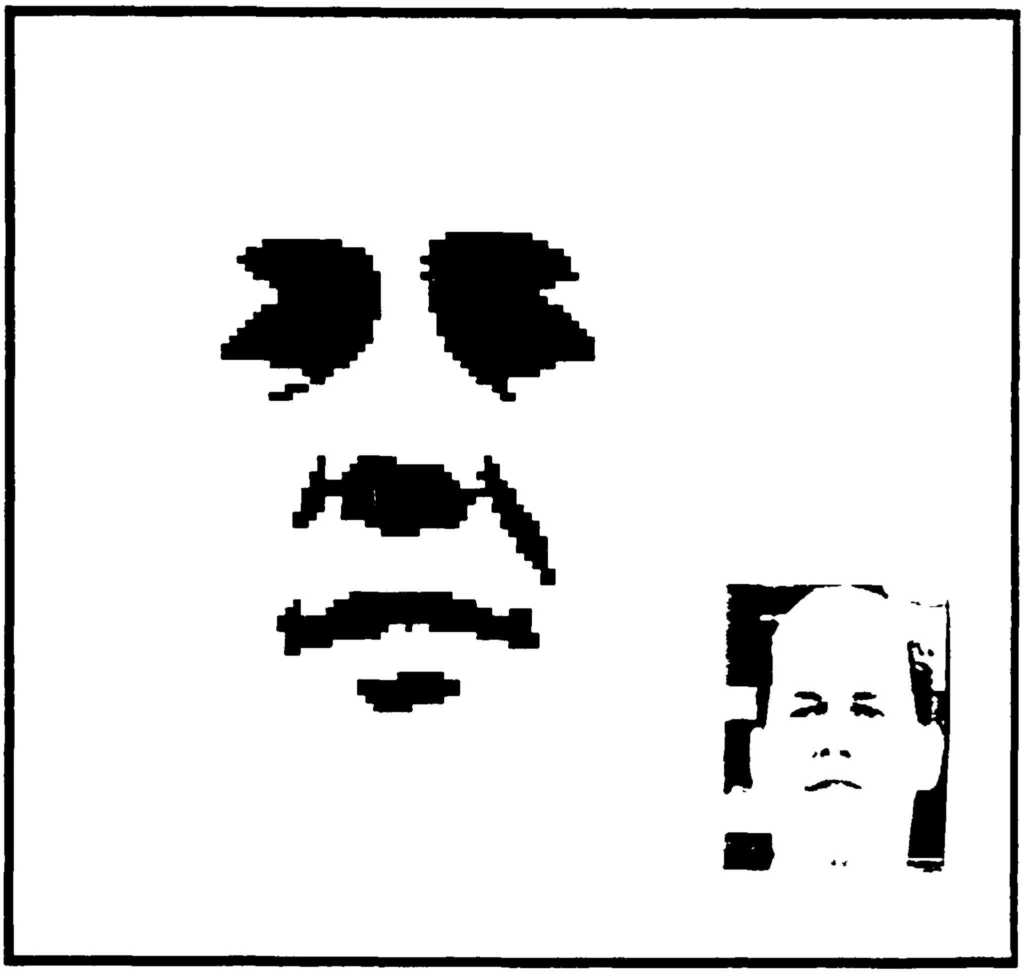


Figure 3-14. Contrast Enhanced Face

Face Location

Part of the scope of this thesis is to evaluate the automatic face location algorithm and attempt to speed up face location while improving recognition accuracy. This section is written in two parts. The first part describes test results from the original face location algorithm and the second part describes a new, two-dimensional, version of the location algorithm.

Evaluation of the Original Algorithm

Smith's face location algorithm was re-written in the C programming language and tested on the Micro-VAX with the following results:

The location algorithm can find all the faces (up to 4) in an input scene in approximately 4 minutes. This includes time to pre-process (smooth) the scene, time to store each face to disk, and time to wipe out each face in the scene (wipe out after saving to disk so it won't be found again).

The face location algorithm is scale invariant. Overall size is not one of the measurements used in evaluating the facial signatures, although a maximum allowable size has to be defined so the facial windows will fit into a reasonable array size for further processing.

The location algorithm is brightness invariant. All the values of minima and maxima are measured in relation to the first maximum found on a signature.

The algorithm is sensitive to variations signatures due to eyeglasses (dark-rimmed) and mustaches. Eyeglasses add

minima to the eye signature and mustaches eliminate the bright region that allows separation of the nose and mouth.

The algorithm is sensitive to head rotation and lighting direction. This is not a problem as long as the subject is looking squarely at the camera and the lighting is directly overhead (two assumptions used in this thesis). However, in testing scale invariance and testing against various backgrounds, subjects had to be positioned in various locations in the lab. This made it impossible to control lighting direction and head positioning. To find out why the AFRM was failing to find many faces that looked like acceptable inputs (in the author's opinion, the face was looking straight at the camera and lighting was overhead) a signature graphing program was developed. This program, called GRAPH.C in Appendix B, allowed the user to plot the brightness variation along any line in the input scene so that measurements could be taken on the eye signatures. The problem with slight changes in lighting and rotation is that the symmetry of the eye signature is destroyed. Allowing greater variations in the measurements made the face locator capable of finding more faces but also increased the number of false alarms. There is no clear dividing line between faces and non-faces using this face location algorithm.

The algorithm was prone to an unacceptable class of false alarms. The signature technique ensures that certain brightness variations are present in a scene before declaring that a face is present. These variations include finding two dark

objects side by side on a light background (eyes) and two more dark objects below and between the first two (nose and mouth). The problem is that the signatures look on a single line through the objects and can't tell what brightness is present above or below this line. Figure 3-15 shows an example of an object that can pass all the measurements of the face locator but does not at all resemble a face. The signatures that were found by the location algorithm have been highlighted in the top of this figure and the "face" has been circled in the bottom of the figure. In this case the scene is random lines drawn on a piece of paper. Other false alarms have included books in a bookcase and a computer screen with several reflections on it.

New Algorithm

Figure 3-15 is an unacceptable input to send to the face recognition algorithm. If there are going to be false alarms now and then, they should at least resemble faces. In an effort to correct this problem, a new face location algorithm has been written that looks at faces in two dimensions (2-D). Instead of looking for dark points on a 1-D line, the new face locator looks for dark objects in a 2-D scene. This new algorithm does have occasional false alarms but the objects it finds always look like faces (when shown to a human subject, the subject can see the "face"). The new algorithm also retains the scale and brightness invarience of the signature based algorithm, is less sensitive to variations in lighting direction and head rotations, and is faster. The

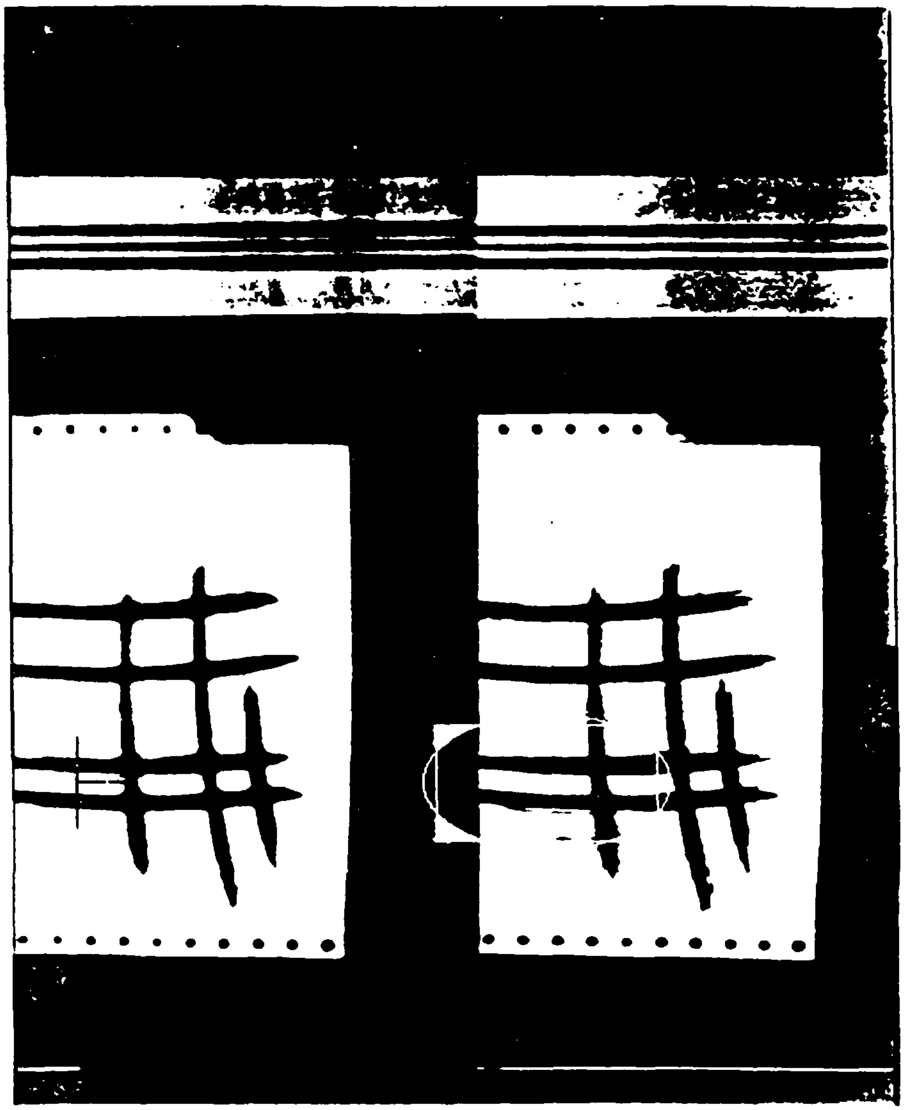


Figure 3-15. An Unacceptable False Alarm

original face locator was removed from the AFRM and put in a program called FACE_SIG.C and is listed in Appendix B.

The new face location algorithm uses binary scenes like the one shown in Figure 3-12. If it finds two dark objects with nearly the same size, one next to the other, then a possible pair of eyes is detected. A "dark" object in this case is an object with a light area all the way around it. This eliminates false alarms in scenes like Figure 3-15. The only task the face location algorithm has is checking for two dark objects (nose, mouth) below and between two others that are side by side (eyes).

In order to help the locator check for a set of features that make up a face, a feature finder was written that looks at all the dark objects in the scene and generates lists of possible eyes, noses and mouths. These lists are passed to the locator which tries to assemble as many faces as possible from them.

The feature finder ensures that the dark objects meet three requirements. The first is that the object is a solid area of dark pixels. The second is that the size of the object is under the maximum size allowed. And the third requirement is that the object can have a block drawn around it that will not touch another dark object (ensures a light area all the way around the object). Figure 3-16 shows a scene after the feature finder was run. All the blocked in objects are possible facial features.

When objects are located and sorted into lists by type,

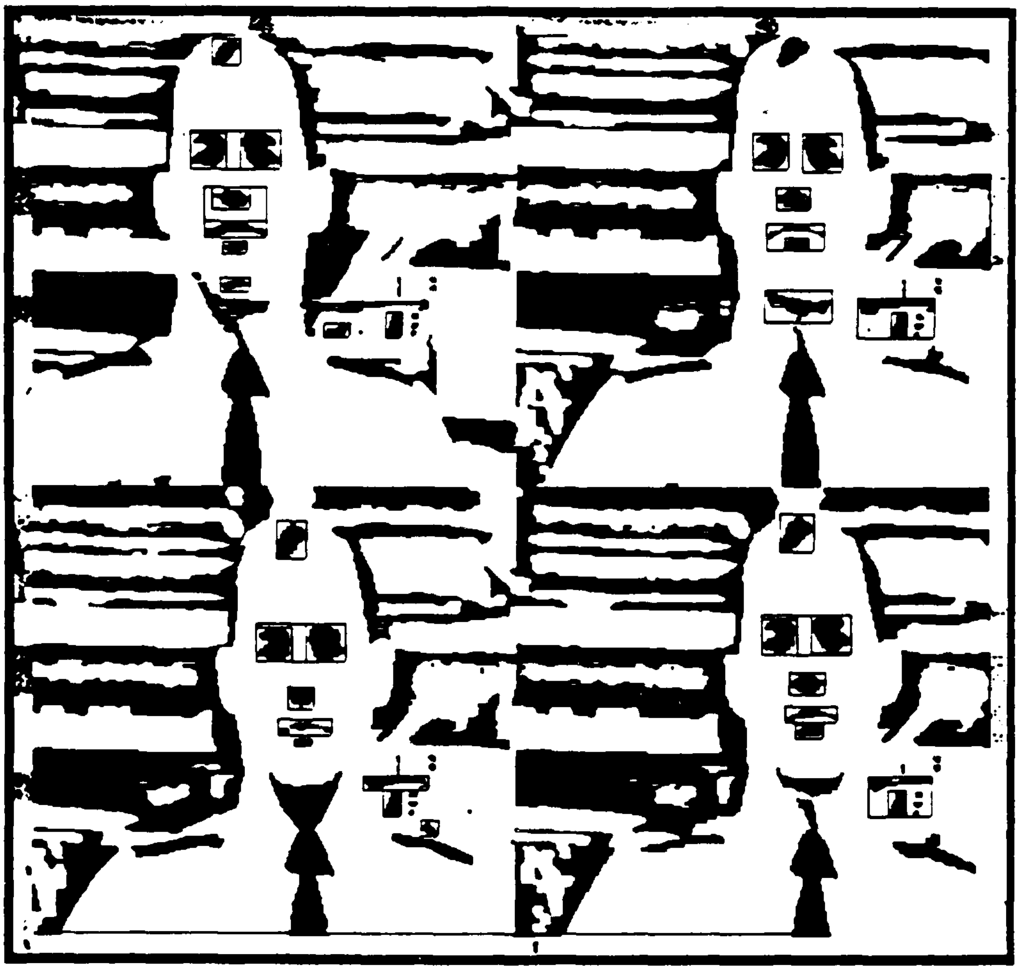


Figure 3-16. Result of Facial Feature Location

their sizes and locations are stored in the feature lists. Then when the face locator determines that a set of features make up a face, all of the feature locations used for windowing the face are already known.

When the new face location algorithm was tested on the Micro-VAX, the following results were observed:

The location algorithm finds all faces in an input scene in less than 2 minutes. This includes time to pre-process (obtain Figure 3-12), time to locate features, and time to store each face to disk.

The location algorithm is scale invariant. As long as feature sizes match each other within a face, it doesn't matter what the overall sizes are as long as they are under a maximum defined limitation.

The location algorithm is brightness invariant. The brightness normalization algorithm takes care of variations in the input scenes.

The algorithm is sensitive to eyeglasses and mustaches if they get in the way of separating facial features.

The algorithm allows slight head rotation and variations in lighting direction (as long as lighting is still from somewhere above the subject). There are no measurements for symmetry between the eyes.

The algorithm will find faces where no human faces exist but the false faces will have much more "faceness" in them. Figure 3-17 shows an example of a false alarm. The top of the figure shows the original scene and the bottom shows a

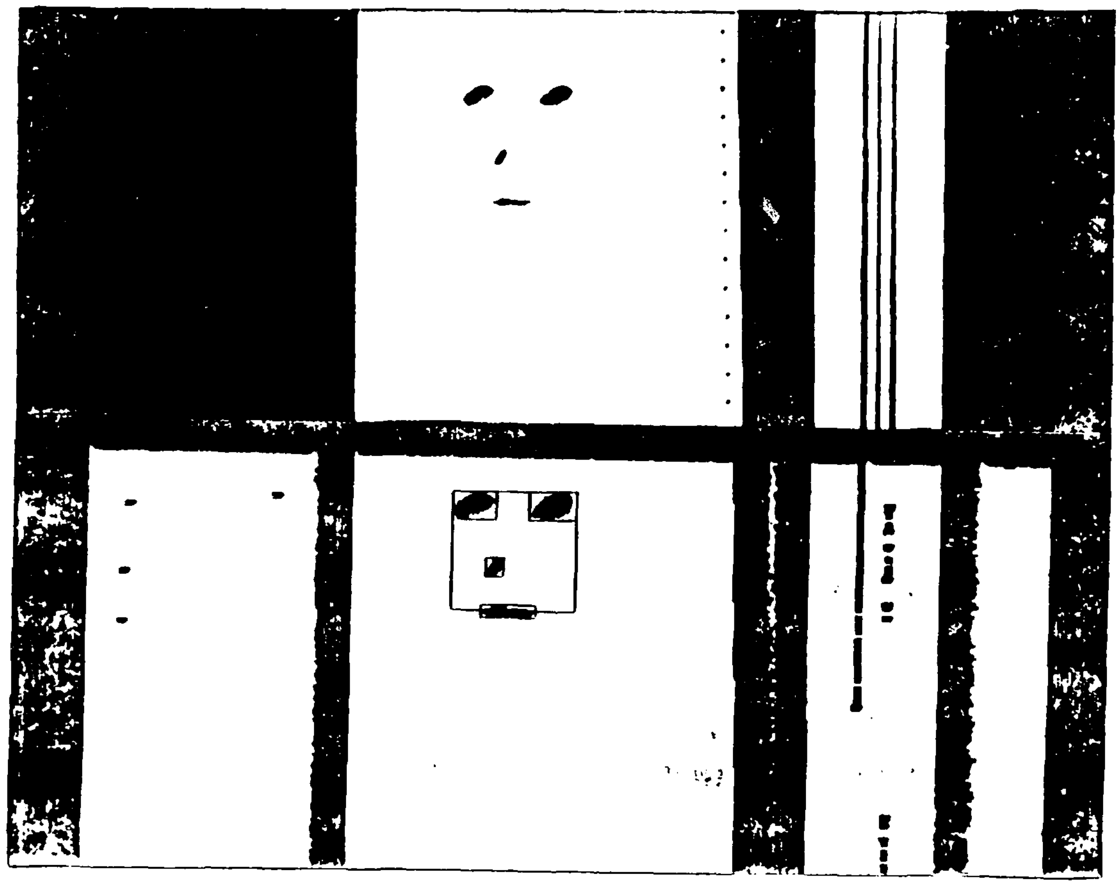


Figure 3-17. An Acceptable False Alarm

block drawn around the internal feature area of the "face" that the location algorithm found. When the algorithm was given the scene in Figure 3-15, it did not find a face.

The new location algorithm is a more reasonable model of a human. It is hard for a human to find a face in a scene given one line of data at a time, but easy if given small regions of the scene. Research into eye scanning patterns (Luria, 1966:467-484) show that it is likely that people are evaluating small areas in a scene. If the brain can determine whether a dark object is present in a small area of a scene, and remember where that area is in relation to all other areas, then it can easily recognize more complicated objects in the scene given some simple rules. For example, a face decision rule would be:

```
If there is a dark object {
  If there is a second one beside it {
    If there is a third below and between them {
      If there is a fourth directly below the third {
        Then there is a face
      }
    }
  }
}
```

The face locator is two subroutines in the program FACE.C in Appendix B. The first is called FACEMAP and it has the decision rules as shown above. It calls the second routine, called FEATUREMAP, which lists all the facial features it can find in the input scene. FACEMAP uses the information in the feature lists to locate and list all the faces in the scene. Along with each facial location, FACEMAP stores feature edge locations that will be needed to window the face.

Windows

When a face is located, it is stored to disk and information about its feature locations are stored in an array. Figure 3-18 shows all the measurements that are stored for the face. All the measurements are taken using the top corner location of the face as a reference so when the face is displayed on another area of the monitor, the measurements remain valid.

The measurements in Figure 3-18 allow the face to be partitioned into the six windows shown in Figure 3-19. The windows are defined as follows:

1. Left Half of Head
2. Right Half of Head
3. Top Half to Nose
4. Internal Feature Area
5. Left Half, Internal Features
6. Bottom Half, Nose to Chin

These windows were selected based on combining the best results of Smith's and Russel's windows as follows:

1. Windows 1 and 2 solve the symmetry problem discussed in Chapter 2. Using the elliptical mask technique should cause these windows to look more like Russel's windows than Smith's.

2. Window 3 yeilded high scores as shown in Figure 2-8. Smith could not use this window because he had only internal feature information available. Russel's window 6 (a similar window) yeilded good results (Russel, 1985:6-11).

3. Windows 4 and 6. Smith And Russel used only one side of the face for these windows so they would not be symmetrical. They lost information however by placing the windows into the corner of an array prior to calculating a gestalt. What they lost was the positional relationship between features on the face. This will be discussed more in the following section.

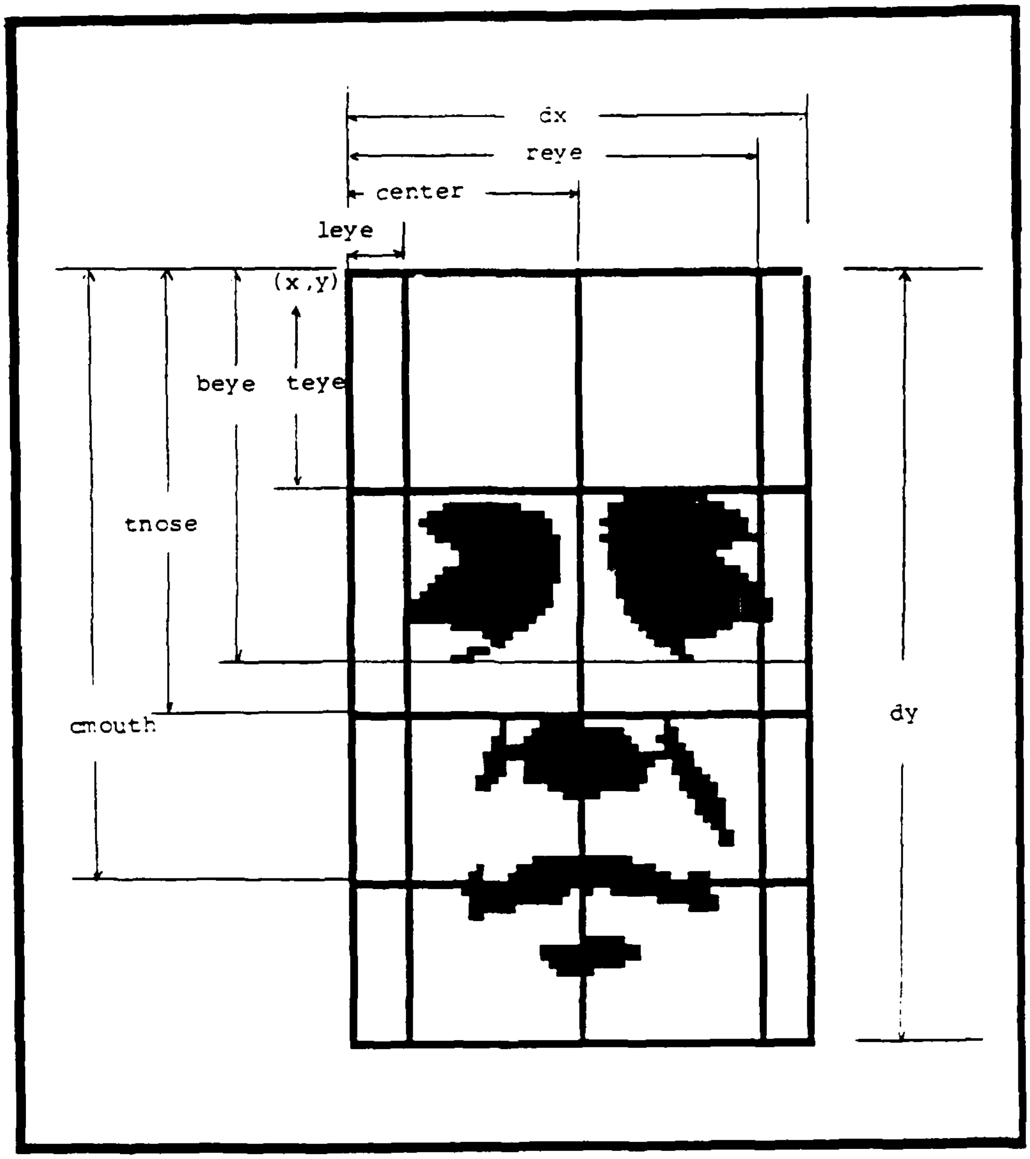


Figure 3-18. Measurements of Facial Features

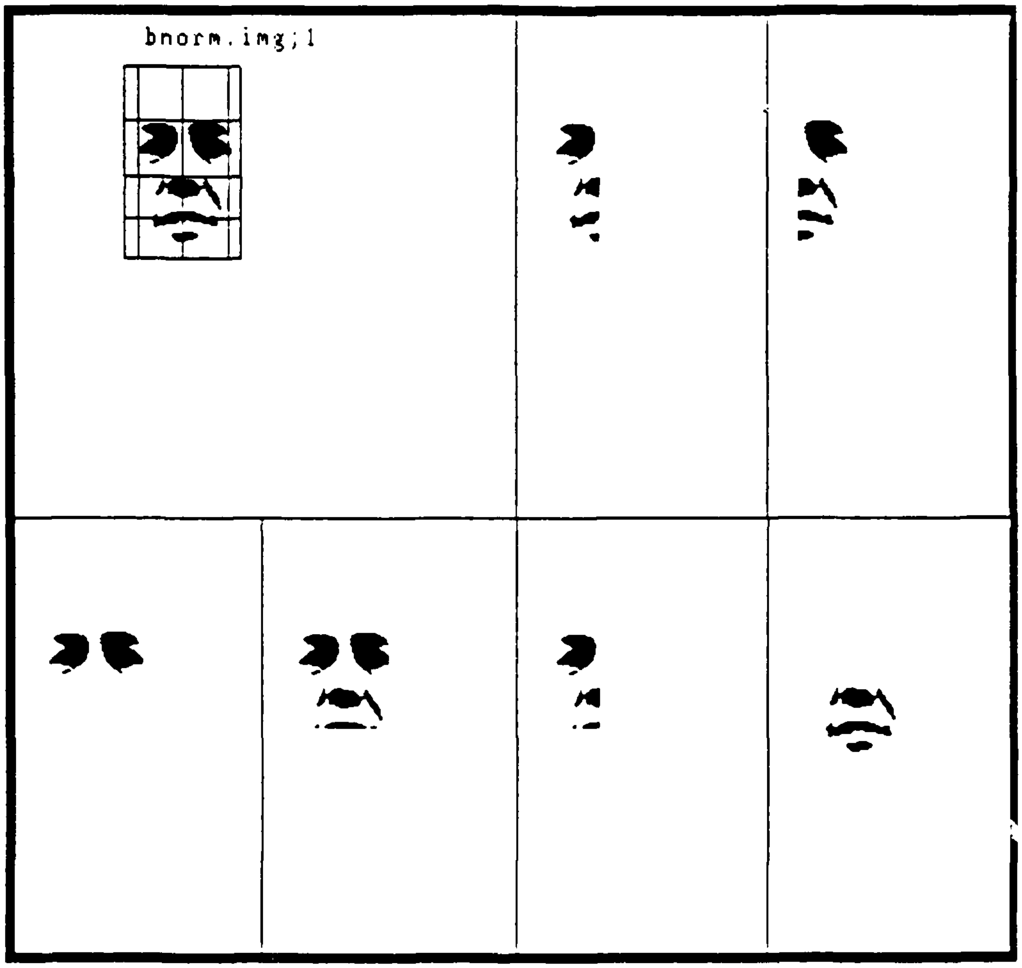


Figure 3-19. New Window Set

4. Window 5 was used by both Smith and Russel with good results. Russel's windows 1, 2, and 4 were his highest performance windows (Russel, 1985:6-11). The present windows 1, 2, and 5 match Russel's windows.

Gestalt Calculation

The gestalt calculation has not been changed from the original algorithm on the Data General system however, three enhancements were made. The first was to speed up the calculation, the second was to increase the resolution of the results, and the third was to increase the separation of faces in the recognition database.

To speed up the gestalt calculation, the size of each window was provided to the subroutine responsible for performing the calculation, CORTRAN16. This subroutine feeds one line (row or column) of the window at a time into the 1-D transform subroutine, RTRANSB. Instead of feeding all lines of the window into the 1-D transform, CORTRAN16 now stops with the last line that contains scene information. So if the window size is smaller than the size of the gestalt array, fewer calculations need to be performed. Figures 2-7, 2-9 and 3-20 all show cases where the window information does not fill the gestalt arrays. The gestalt calculation for all six windows of a face now takes less than 3 minutes to complete, compared to Russel's 8 minutes (Russel, 1985:5-51).

To increase the resolution of the gestalt calculation, the gestalt array (where the window information is put) was

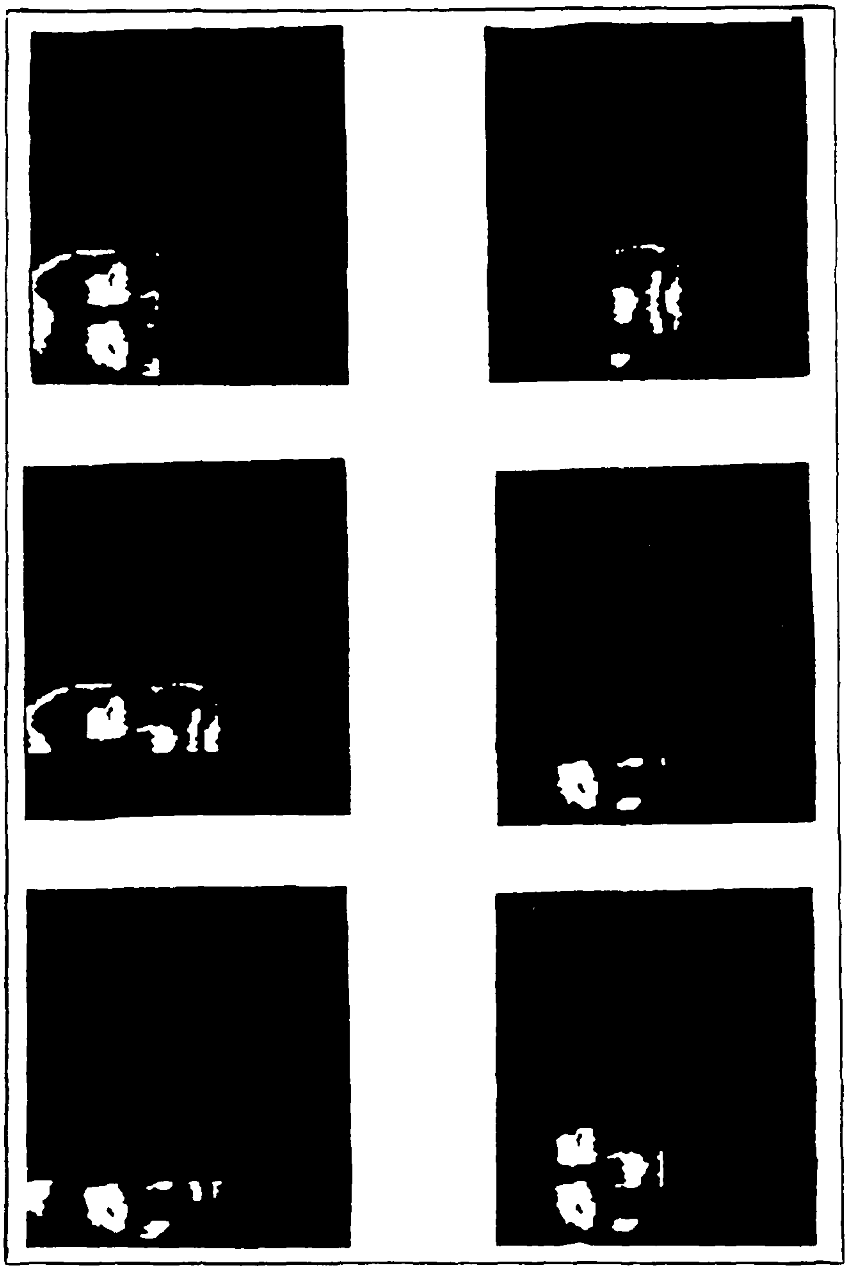


Figure 3-20. Window Placement in Gestalt Array

increased from 64X64 pixels to 128X128 pixels. This also allows the recognizer to handle faces larger than 64X64, the previous size limit (Smith, 1986:5-8).

To increase the separation of faces in the recognition data base (increasing the accuracy) an idea suggested by Russel was tried in the AFRM (Russel 1987). This idea was to maintain the positional relationships between features on the face by placing the windows into the gestalt array in the position that they appear on the face. Figure 3-20 shows how the windows are now placed in the array (no longer in the upper left corner). Now instead of a gestalt value being referenced to the corner of the window, the gestalt value for each window is referenced to the corner of the whole face from which it came. To see if there was any difference in the separation of faces, the gestalt values were calculated for a set of six faces (3 of subject LL and 3 of subject JS) with both window placement techniques. Table 3-1 shows the average gestalt values obtained for each subject. Comparing the separation of the average gestalt values for these subjects resulted in Table 3-2.

The separation of gestalt values was better for all the windows, when the windows were placed in the gestalt array based on facial placement, so this technique was used. One additional change was required to make this technique work. In order to scale the gestalt values to a standard size face the scale factor had to be calculated using facial size instead of window size as shown in Figure 2-12.

Table 3-1. Gestalt Values for 2 Window Placement Techniques

#	Windows Placed in Corner of Array LL(X,Y)	Windows Placed in Corner of Array JS(X,Y)	Windows Placed by Facial Location LL(X,Y)	Windows Placed by Facial Location JS(X,Y)
1	21,50	26,58	23,51	27,60
2	12,56	17,67	47,59	59,69
3	38,36	44,45	36,39	43,50
4	39,54	45,54	37,59	44,59
5	25,51	29,52	23,56	27,58
6	39,95	49,94	37,100	48,100

Table 3-2. Separation of Gestalt Values

Window #	Best Separation with Placement Referenced to corner of:
1	Either
2	Face
3	Either
4	Face
5	Face
6	Face

Recognition

The algorithms described up to this point have been designed to re-create the ideal image acquisition conditions that Russel had in 1985, but with no human intervention allowed. Russel had fixed lighting and background conditions, a fixed setting on the camera, and operator input was allowed (which provided the AFRM with the exact edges of the head).

The AFRM had high recognition scores when it was given faces that met all of Russel's conditions. Russel's results are shown in Table 2-1. When conditions were allowed to vary, the recognition accuray dropped, as shown in Table 3-3.

Table 3-3. Smith's Test Results (Smith, 1986:5-14)

Number in Database	: 20
Number Recognized as 1st Choice	: 12
Number Recognized as 2nd Choice	: 4
Number Recognized as 3rd Choice	: 2
Number Recognized as 5th Choice	: 1
Number Recognized as 8th Choice	: 1
Absolute Correctness	= 0.60
Average Reduction in Uncertainty	= 0.9525

The new preprocessing, location, and windowing algorithms should cause an increase in accuracy when used with the original recognition algorithm, so Russel's algorithm "REMID" was implemented as the subroutine, "RECOGNIZE" in FACE.C, and tested. The distance measurement used in RECOGNIZE for each window (w) is shown below (Russel, 1985:4-40a) and is multiplied by a performance factor for the window (w).

$$v[id][w] = \exp\left\{ \frac{-1}{1.4} \left[\frac{(gix-gux)^2}{(2*sigix)^2} + \frac{(giy-guy)^2}{(2*sigiy)^2} \right] \right\}$$

where v[id][w] = the distance from candidate (id) to the unknown individual for window (w).

gix,giy = X,Y coordinate values of previously stored candidate (id).

gux,guy = X,Y coordinate values for an unidentified individual.

sigix,sigiy = X,Y standard deviations for person (id).

The recognition algorithm was tested with and without window performance factors and the results are given in Chapter 5. Figure 3-21 shows a typical output from the recognizer where the top image is the person to be recognized, and the bottom images are pictures of the three closest individuals in the database.

Summary

This chapter presented the changes made to the AFRM as part of this thesis effort. Discussion of changes was kept at the algorithmic level and was kept as independent of implementation as possible. Chapter 4 will describe how all the algorithms were gathered together into a complete AFRM program and implemented on the Micro-VAX II computer.

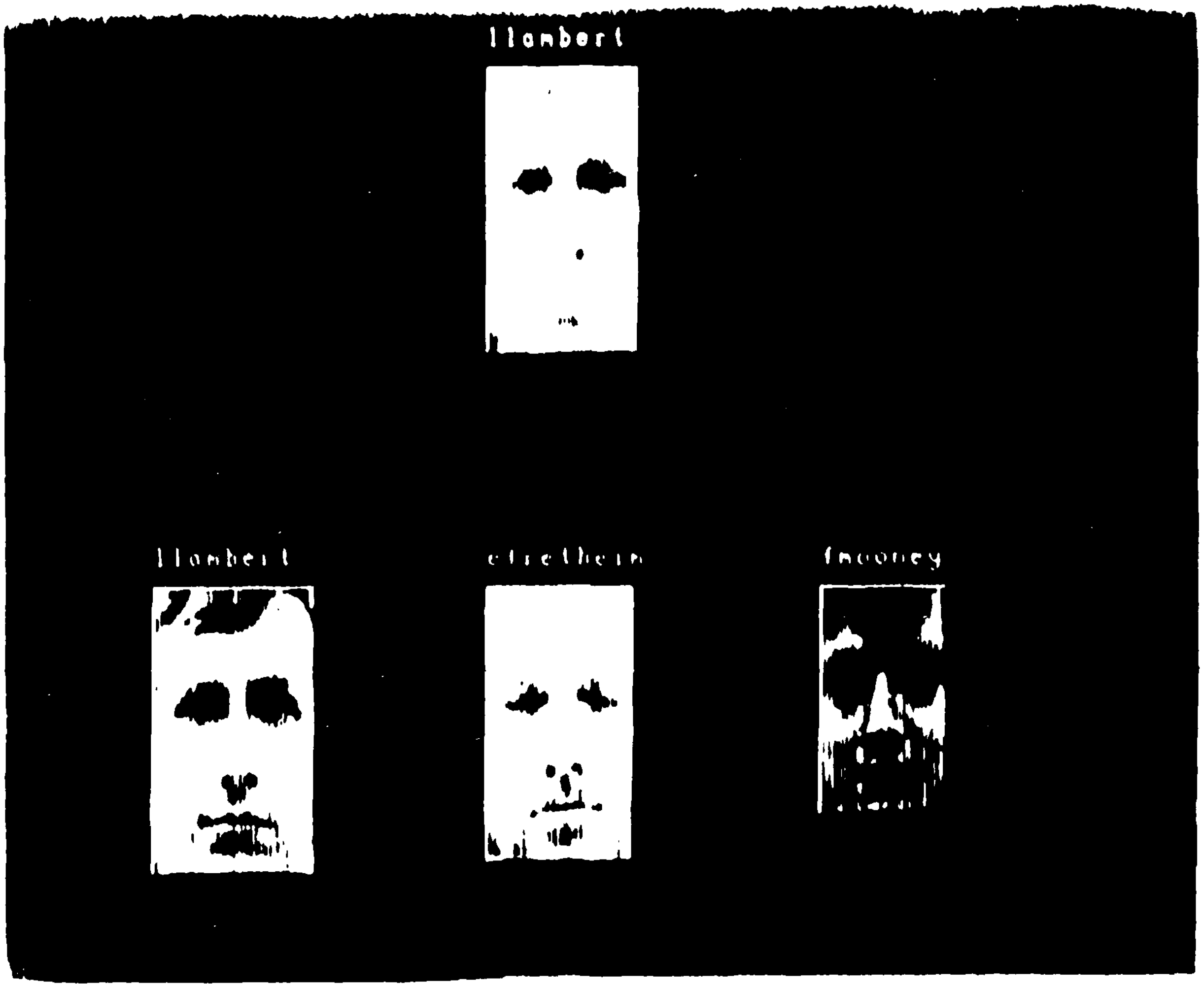


Figure 3-21. Output From RECOGNIZE

IV. Implementation

The first section of Chapter 3 discussed the need for a new environment for the AFRM. The Data General system could not adequately provide the speed and memory needed by the AFRM so a Micro-VAX II was chosen as the new host computer. This chapter discusses the details of implementing the AFRM on the Micro-VAX II designated SMV2A. Operational details are provided by the User's Manual given in Appendix C.

Software

Software for the AFRM was written in VAX C version 2.0. The software has to be linked to several libraries including the Imaging Technology (ITEX) library of image processing subroutines. Appendix C describes the linking requirements in detail. Appendix B contains the source code for the AFRM and a list of the required ITEX subroutines.

The Micro-VAX uses the MicroVMS V4.2 operating system, however the AFRM code can be easily modified for use on other systems. There are only two direct references to the operating system by the AFRM code. The first is a call for the operating system to delete files using the "system()" command (Kernighan and Ritchie, 1978:157). The second is a reference to directories and subdirectories where files are stored, and the directory structure is dependent on the operating system.

The AFRM is located in a directory on the SMV2A Micro-VAX called FACE. It can be accessed by logging on with username "FACE" (no password is required). The User's Manual describes how the AFRM is protected against accidental change and erasure and how the protection can be removed in order to modify the software. The AFRM database files are located in a subdirectory called "FACE.DBASE".

Program Structure

The source code for the AFRM, called FACE.C, contains all the algorithms described in Chapter 3, all the code necessary for maintaining a facial database, and the menu structure shown in Figure 4-1. When a user logs on to SMV2A (as FACE), the main menu is displayed on the computer screen. This menu provides access to the individual AFRM algorithms and access to the database and demonstration options that will be described in this chapter. Descriptions of all the menu options are given in the User's Manual in Appendix C.

The AFRM menu structure is set up so that a user can run each algorithm independently of all the others. In addition, an option called "Total System" is provided that runs all the algorithms in series as shown in Figure 4-2. No operator inputs are required once this option is selected except for selecting which camera to use and when to quit. Figure 4-2 shows the time spent in each algorithm and the time required for each loop. The subroutine in FACE.C that is represented by this flowchart is called "AFRM()".

** Main Menu **

1: Acquire Images
2: Find Faces
3: Gestalt and Identify / Save

4: Display Contents of Dbase
5: Delete a Subject
6: Delete an Image
7: Train

8: Demonstration

0: Quit

** Acquisition of Images **

0: Return to Main Menu

1: Stationary Target
2: Moving Target
3: Load Image from Memory
4: Save Image in [FACE]

5: Set Camera Port
6: Camera Check
7: Re-Initialize Hardware

** Demonstration **

0: Return to Main Menu

1: Identify a Person
2: Total System

Figure 4-1. AFRM Menu Structure

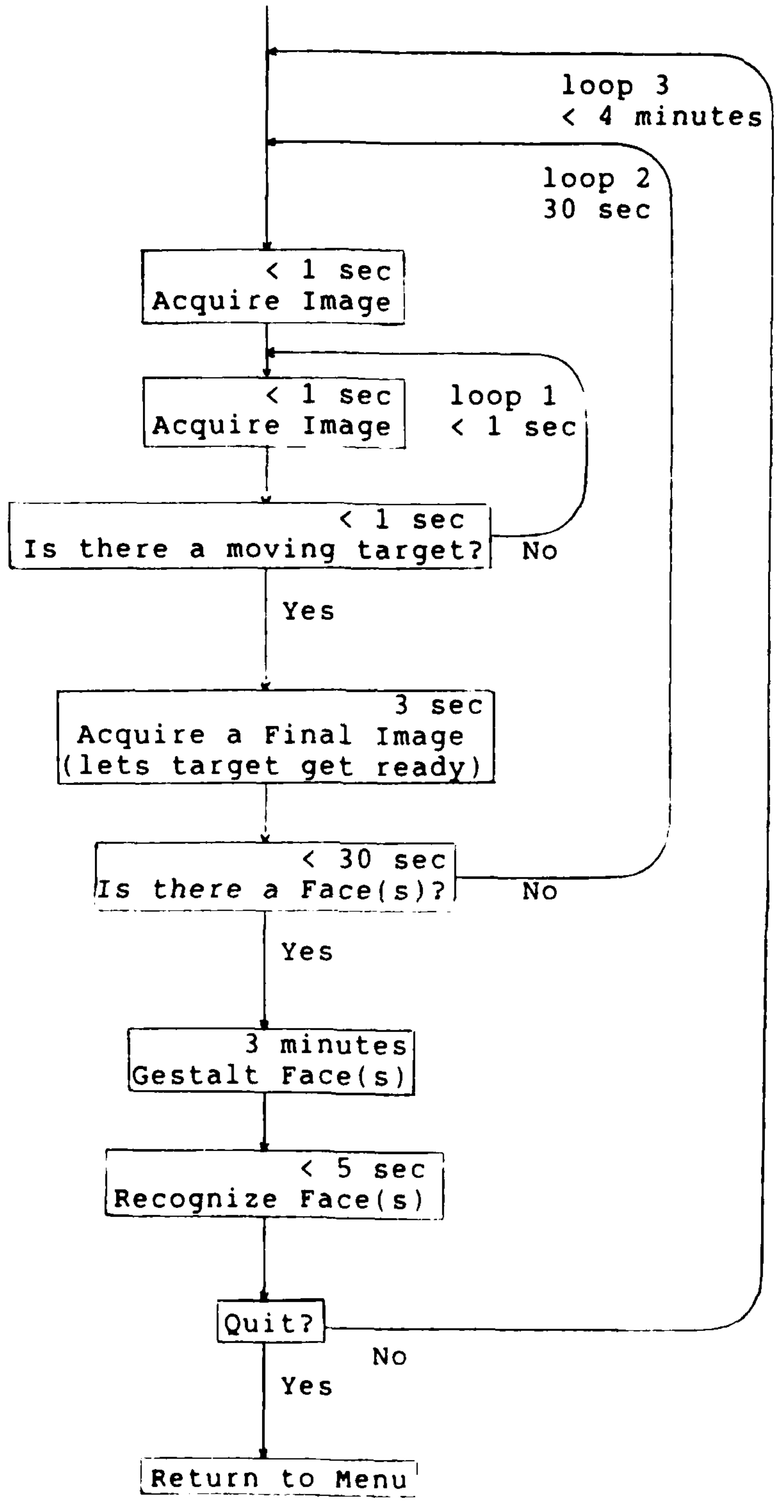


Figure 4-2. Total System Option

Database Design

A database was written in order to maintain the gestalt files of all subjects entered into the AFRM. This includes the files for subjects that the AFRM is trained with, and extra files used for testing the recognition capabilities of the AFRM. When the AFRM is not running, all the gestalt data are kept in two files called TRAIN.DAT;1 and OTHERS.DAT;1. These files are ASCII text files, so they can be displayed on the computer screen and printed to a printer. Examples of these files are shown in Appendix D. The database software is contained in subroutine "MENU1" and is supported by the following subroutines which are all contained in FACE.C:

COPYFILE	GETINT
DEL	READFILE
DISPLAY	WRITEFILE

When the AFRM is first activated (by logging on), the two disk files are read into arrays in memory. The contents of these arrays are modified using Main Menu options 3 through 7 while the AFRM is running and when option 0 (Quit) is selected, any modifications are written to disk. It is important that the AFRM is terminated only by using the Quit option, because all other methods of terminating the program (CTRL-C, CTRL-Y, etc) will prevent the modified array data from being stored to disk. The method used to prevent this potential problem is discussed in the User's Manual.

The arrays used to hold gestalt data are implemented using C Structures so that subject names, file versions and gestalt values can all be held in the same array. The two

structures are called TLIST and ILIST and the contents of these structures are global, available to all the subroutines that need to access them. Main Menu options allow the user to add to (by training), subtract from (delete trained subjects or individual images), and display the contents of the structures.

The gestalt data files are stored on disk in a directory called [FACE.DBASE] along with all the picture files. The picture files are stored by subject name and an extension that identifies the status of the subject: IMG files are pictures that the AFRM has not been trained with, and PIC files are pictures the AFRM has been trained with. These extensions are automatically changed by the AFRM when a subject is moved from one area of the database to the other. The picture files are also stored with version numbers that match the version numbers in the data files, so the gestalt file is always linked to the photo it was obtained from.

Summary

This chapter has described how the AFRM was implemented on a Micro-VAX II. Additional information can be found in the equipment list in Appendix A and the user's manual in Appendix C. A more detailed description of the AFRM code can be found in the comments within the source code given in Appendix B.

V. Test Results

This chapter presents the results of testing the AFRM face location and recognition capabilities, and discusses what these results mean.

Effect of Camera Settings on Performance

The brightness normalization algorithm presented in Chapter 3 converts all scenes that are input into the AFRM, into normalized scenes that all have the same brightness level, or DC term. Since all brightness variations are now always around the same center value (128), it is easy to decide which areas are "dark" and which are "light" in a scene. The brightness normalization algorithm allows the AFRM to process scenes with a wide range of lighting conditions and allows the operator to use a wide range of camera f-stop settings.

The gestalt calculation used in the AFRM is basically a center of mass calculation where dark regions on the face have more "mass" than light regions. The ability to locate a face is dependent on finding the high "mass" areas of the face in a consistent manner. These areas do not change much when the camera is out of focus. They do start blending in with the rest of the face but they can be located in very blurred scenes. Since the location of facial features does not change with camera focus, faces can still be recognized.

The gestalt calculation is also able to handle faces of different size. By scaling all faces to the same size, or in this case, scaling their gestalts based on a standard sized face, the recognition results become independent of scale. The location algorithm is independent of scale by allowing it to look for variable sized features and requiring it to assemble faces out of features that match each other in size. The Dage camera has a zoom lens which affects the size of the faces entered into the AFRM.

To verify the ability of the AFRM to deal with the variables discussed above (f-stop, focus, and zoom), a test was performed in which each variable was set three times and everything else was held constant. The results of the test are shown in Table 5-1.

One constant was difficult to acheive and it is assumed that most of the variation in gestalt data is due to this problem. This "constant" was the subject in the input scene. Since the face used for this test had to be held constant, a subject was asked to sit as still as possible for several minutes while a series of pictures was taken on video tape.

Table 5-1 shows the three variables in the first column and the image version number assigned by the AFRM in the second column. .n all, there were ten images entered into the AFRM. The AFRM was not trained with the subject prior to the test. The last two columns show the distance to the closest candidate in the database and the candidate's name.

The AFRM was trained with the first four images entered during the test and thereafter the AFRM recognized the subject as "fmooney", the correct result. The candidates "mkabrisky" and "bgeorge" continued coming up, but as the second choice after the AFRM was trained.

The AFRM started assigning version numbers from 1 after the AFRM was trained with the first four because once an image is used for training, it receives a different file name (It is a .PIC file, no longer .IMG). At the beginning of the test the AFRM was trained with fourteen subjects. At the end of the test, image files for all the subjects were tested to ensure that the AFRM still recognized them correctly (that nobody else was recognized as fmooney).

Variable		#	Win 1	Win 2	Win 3	Win 4	Win 5	Win 6	#1Dist	Candidate
	5.6	1	25,54	45,54	36,38	36,61	22,58	38,101	.358	mmayo
F-	8	2	22,62	44,71	33,51	33,62	22,60	35,97	.603	mkabrisky
Stop	11	3	22,58	47,63	38,45	36,65	22,63	40,97	.544	bgeorge
	16*	4	23,64	46,64	37,50	37,64	23,62	39,98	.503	bgeorge
		1	22,64	44,68	33,53	33,62	22,60	38,97	.659	fmooney
Focus		2	22,65	43,70	34,53	34,63	22,60	36,94	.639	fmooney
		3	24,57	44,60	37,44	37,63	24,60	39,97	.736	fmooney
		4	24,62	46,66	35,51	35,62	24,60	38,97	.758	fmooney
Zoom		5	23,64	47,73	35,55	35,64	23,61	38,96	.695	fmooney
		6	23,58	50,63	36,47	38,63	23,61	40,97	.718	fmooney

* At this point, train with 1st 4 images and set F-Stop = 8. All images from here on should be recognized as fmooney.

Table 5-1. Effects of F-Stop, Focus and Zoom on Location and Recognition Performance

False Alarms and Missed Faces

The new face location algorithm was compared to the original, signature-based, algorithm in Chapter 3 and was found to be less prone to false alarms (finding faces where none exist). This result is good as long as the location algorithm doesn't start missing real faces in order to keep the false alarm rate down. To see if this was occuring, the two location algorithms were tested on a set of twenty scenes shown in Appendix F. Ten scenes had faces present and ten did not. The scenes covered a large range of backgrounds, lighting conditions and scale. Table 5-2 shows the results of the test for each scene. Table 5-3 summarizes the results and clearly shows that the new face location algorithm, AFRM, performs better than the original algorithm, FACE_SIG.

	Scenes With Faces			Without Faces	
	FACE SIG	AFRM		FACE SIG	AFRM
1	F,A	F	11	N	N
2	N,A	F,A	12	N	N
3	N	N	13	A	N
4	F	N	14	N	N
5	F	N	15	N	N
6	N	F	16	N	N
7	F	F	17	A,A	N
8	N	F	18	A	N
9	F	F	19	N	N
10	F	F	20	A	N

[F=Face, A=False Alarm, N=No Face]

Table 5-2. Face Location Test

	FACE SIG	AFRM
False Alarms (20 scenes)	7	1
Faces Found (10 faces)	6	7

Table 5-3. Summary of Face Location Test

Recognition Score

Several tests were performed to measure the recognition capability of the AFRM. This capability is measured using two scores. The first score is the percentage of time the AFRM is absolutely correct in recognition (the list of candidates put out by the AFRM in response to an unknown input has the proper answer in the top position). The second score is the percentage reduction in uncertainty. This score, developed by Russel gives an indication of how good the AFRM is, when it is not absolutely correct (Russel, 1985:6-8). It provides an indication of where the correct answer is in a list of candidates (for example, it might always be in the top 3 in a list of 20). For all the tests, the AFRM was trained with four pictures each of ten subjects. A fifth picture of each subject was used for testing.

The first test was to measure the recognition performance obtained by using a single window from the face. This was done for each window and the results are shown in Table 5-4. This table shows where the AFRM placed the correct answer in its ordered list of candidates. For example, using window 1 it placed the correct answer for a picture of llambert in the fifth position.

Table 5-4 shows that some windows performed better than others in this test. In this case, window 4 alone was enough to correctly identify all ten individuals in the database. This does not mean that all the other windows can be discarded however, because they do provide useful information.

As the database grows, window 4 alone will not be sufficient. Also, a combination of the data from all six windows provides the same recognition result (100% correct) but with a higher confidence level.

Input Photo	Window # 1	2	3	4	5	6
llambert	5	1	1	1	1	3
efretheim	2	2	1	1	2	1
mkabrisky	2	1	4	1	3	2
ecrawford	4	3	2	1	2	3
mdrylie	1	2	2	1	1	1
mmayo	3	3	2	1	1	3
mlambert	1	1	1	1	1	1
jsillart	1	1	1	1	1	2
dlambert	1	1	1	1	2	1
gdawson	1	2	1	1	3	1
% absolute correct	.50	.50	.60	1.0	.50	.50
% reduction in uncertainty	.89	.93	.94	1.0	.93	.89

Table 5-4. Recognition Performance for Single Windows

Confidence Level

The second test was performed to get an indication of the confidence that should be placed on the results. Confidence should be based on two factors, the closeness of the unknown photo to a given candidate's data, and the difference in distance to this candidate and all others. Chapter 3 shows that distance, as measured by the AFRM, varies from 1.0 (a perfect match) to 0.0 (no similarity at all). An example of calculating confidence level follows.

Suppose the AFRM is to recognize a photo of llambert and it puts out the following list of candidates:

1. llambert distance = .93
2. mkabrisky distance = .92
3. srogers distance = .11

It is clear that the llambert file closely matches the input photo, but so does the mkabrisky file. The srogers file is quite far from the input photo. The confidence level assigned to both candidates 1 and 2 should be nearly the same (about 50%). Now suppose the AFRM has put out the following list:

1. llambert distance = .49
2. mkabrisky distance = .01
3. bgeorge distance = .002

If it is assumed that the AFRM was trained on llambert, then the AFRM is correct. The second closest candidate is much farther away than the first candidate so candidate 1 should be accepted with a high level of confidence (maybe 90%). But candidate 1 is also a significant distance away from the unknown photo too. What if the input photo does not represent anyone in the database and a distance = .49 comes out for llambert? To cover this situation, the magnitude of the distance must be considered in the confidence calculation.

This example has shown that a low distance number should reduce the confidence level and a low difference between candidate distances in the ordered list should reduce the confidence level. Table 5-5 shows how the addition of windows can improve confidence level. The table starts with the best window and adds windows (ordered by window performance shown in table 5-4) as shown across the top of the table. Each entry in the table represents the distance to the first and second candidates in the ordered list. In all cases, the first candidate was the correct answer (because the table starts with window 4, it has 100% absoulte correct recognition results).

Input Photo	Windows Used 4	+3	+2	+5	+1	+6
llambert	.7881	.7101	.7514	.7396	.6580	.6626
	.6583	.5068	.5163	.4837	.4546	.4386
efretheim	.9906	.8769	.8018	.7548	.7505	.7380
	.8476	.7458	.7170	.6959	.6190	.5592
mkabrisky	.9826	.6696	.7456	.6435	.6306	.6469
	.7232	.4340	.4509	.3669	.3261	.3558
ecrawford	.7019	.6870	.6806	.6309	.6230	.6241
	.6620	.5168	.5946	.4954	.5218	.5200
mdrylie	.8610	.8978	.8305	.8368	.8242	.8257
	.4374	.4047	.4990	.4135	.4144	.4239
mmayo	.9563	.8741	.6773	.6410	.5813	.5787
	.9270	.7529	.6421	.5931	.5524	.4972
mlambert	.9980	.8655	.8888	.9059	.9157	.8997
	.0009	.0598	.2066	.1678	.1784	.1607
jsillart	.8532	.9119	.8893	.8794	.8806	.8367
	.5552	.3599	.3888	.3258	.3783	.3577
dlambert	.4540	.6059	.6875	.6705	.6179	.6337
	.1031	.1091	.1709	.1760	.1565	.1427
gdawson	.2220	.4512	.5136	.4949	.4891	.4912
	.0797	.1818	.3637	.4717	.4197	.3856

Table 5-5. Distance to First Two Candidates in List

At first glance, the data in Table 5-5 appear to contradict the ideas presented above. In eight out of ten cases, the distance to the correct candidate has decreased by the addition of windows (only dlambert and gdawson distances increase). But the distance to the second candidate has also increased as shown in Table 5-6. This table shows the difference between the pairs of numbers in Table 5-5. This difference increases for seven out of ten of the subjects, increasing the confidence level that should be assigned to the ordering of the candidate list.

Input Photo	Windows Used 4	+3	+2	+5	+1	+6
llambert	.1298	.2043	.2351	.2559	.2034	.2240
efretheim	.1430	.1311	.0848	.0589	.1315	.1788
mkabrisky	.2594	.2356	.2947	.2766	.3045	.2911
ecrawford	.0399	.1702	.0860	.1355	.1012	.1041
mdrylie	.4236	.4931	.3315	.4233	.4098	.4018
mmayo	.0293	.1212	.0352	.0479	.0289	.0815
mlambert	.9979	.8057	.6822	.7381	.7373	.7390
jsillart	.2980	.5520	.5005	.5536	.5023	.4790
dlambert	.3509	.4968	.5166	.4945	.4614	.4910
gdawson	.1423	.2694	.1499	.0232	.0694	.1056

Table 5-6. Difference Between Candidate 1 and 2 Distances

Up to this point, confidence level has been discussed but no values of confidence have been calculated. Russel calculated a probability value to indicate how sure the AFRM was of the candidate ordering, but this probability was based on the similarity between an unknown individual and a candidate (the actual distance number put out for the candidate) only (Russel, 1985:4-43). It was not based on the difference between distances to the rest of the candidates in the list (like numbers found in Table 5-6). In addition, an equation to calculate a confidence value may be dependent upon the number of individuals in the database. Since insufficient data are available at this time, no confidence values have been calculated.

Window Performance Factors

A third test was performed to see if differences between candidates could be increased further by adding window performance factors. Russel assigned higher weighting factors

to the better performing windows before combining window measurements into a final distance value (Russel, 1985:4-43). This gave the better windows more influence in the recognition process. In order to see what effect window performance factors could have on the recognition performance, the window factors were set to fixed values (not determined by contents of database as done in Russel's thesis). Table 5-7 shows the two sets of performance factors tried. The second set gives more weighting to the windows determined as best by Table 5-4.

Window #	Set 1	Set 2
1	1.0	0.6
2	1.0	0.9
3	1.0	1.2
4	1.0	1.8
5	1.0	0.9
6	1.0	0.6

Table 5-7. Window Performance Factors

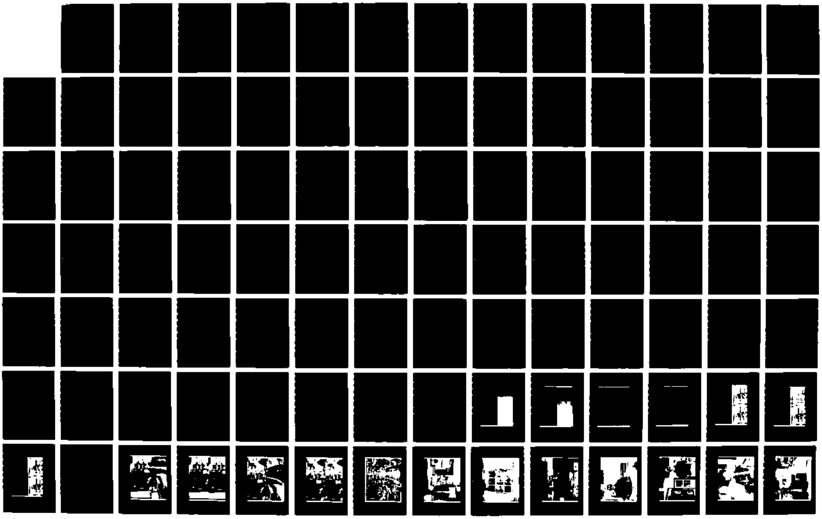
No conclusions can be drawn by the test results because [illegible] separation of candidates increased for five subjects an[illegible] decreased for the other five subjects when set 2 was [illegible] Both sets of window performance factors yeilded [illegible] correct recognition scores. A larger database [illegible] before window performance factors will make a [illegible] the recognition performance.

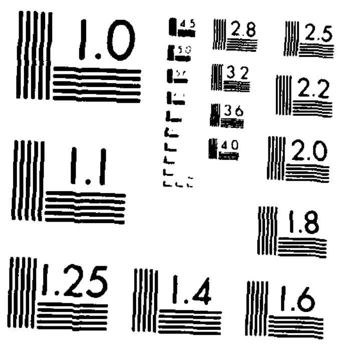
Summary

These test results indicate tha[illegible] at least as well as the previous [illegible]

AD-A188 019 EVALUATION AND ENHANCEMENT OF THE AFIT AUTONOMOUS FACE RECOGNITION MACHINE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING L C LAMBERT DEC 87 AFIT/GE/ENG/87D-35 2/3

UNCLASSIFIED F/G 12/9 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963 A

in all areas. The face location algorithm has been improved with no adverse effect on the recognition scores. The recognition scores appear to be better than the scores shown in Tables 2-1 and 3-3 however there are fewer subjects in the database (presently at 15 subjects and still 100% correct). In addition, all algorithms are faster and the AFRM can process a wider variety of input scenes.

VI. Conclusions and Recommendations

This chapter presents conclusions of this thesis effort and recommendations for further research with the AFRM.

Conclusions

The AFRM is proof of the capability of recognizing a human face using a machine. The AFRM does this recognition quickly and accurately. It has been based upon theories of how the human physiology works, and attempts to model processes that are physically realizable by the eyes and brain. Whether the AFRM incorporates techniques actually used by the human visual system is, of course, unknown since these brain mechanisms remain undescribed at present.

During this thesis effort, the automatic face location and windowing capabilities have been enhanced to the point where the AFRM is completely autonomous. The algorithms have been sped up to the point where real-time processing can take place (it is only seconds between input and output). The AFRM has been successfully implemented on a modern computer with readily available hardware and software. These factors make the AFRM more useful as a research and teaching tool, and pave the way for further understanding of human recognition capabilities.

Recommendations

In addition to the recommendations made by Smith and Russel that have not yet been implemented, the following areas should be explored:

I. To further improve image pre-processing, the following suggestions:

1. Implement processing of color images to increase information available to the recognizer, improve separation of the head from the background, and possibly allow definition of a better facial feature set.
2. Implement processing of images from a pair of cameras, utilizing binocular disparity techniques to separate the head from the background. This may also provide information useful to the recognition process by having available, two images of the subject each taken at the same time but different angles.
3. Implement algorithms in a parallel hardware archetecture, creating a model of the human eye.

II. To improve face location capability:

1. Combine the two face location techniques discussed in this thesis to find more faces while reducing false alarm rate. Table 5-2 shows that a combination of the two techniques could result in 90% of the faces being found, with only a single false alarm.

2. Come up with a better set of facial features to use in place of the dark areas now being used.

III. To improve recognition capability:

1. Study effects of head rotation on recognition score and find a way to overcome the present limitations.
2. Implement an algorithm with a truely constant recognition time. The present algorithm is very fast, but may slow down when the database grows into the hundreds or thousands.
3. Thoroughly exercise the AFRM, training it with many more subjects to find out how it performs and what its limits are. Develop algorithms to overcome these limits.
4. Add more information to the recognizer by improving the quality of the images sent to the gestalt algorithm. This might be done by using the brightness normalized faces without the contrast enhancement.
5. Add more information to the recognition process in the form of voice data, subject's height, eye color, etc.

IV. To improve availability of the AFRM as a research or teaching tool:

1. Implement the AFRM on an IBM PC/AT computer. An Imaging Technology, image processing board exists which would support this effort.

V. To more closely model the human form of face recognition:

1. The AFRM is required to perform recognition on a single, sometimes poor quality image and produce an output list of candidates. There is some difficulty in expressing the confidence level that should be placed on the ordering of this list. Humans too have doubt sometimes in their own recognition capabilities. A human does not usually have to produce an answer from one quick look however. The human has a continuous look-and-update recognition process. Any doubt that exists in the viewers mind causes additional acquisition and processing of data until finally, all doubt is removed.

The AFRM should be given the same chance to acquire additional information that the human has. This could be in the form of multiple images of a subject. The AFRM would first need to be made as fast as possible to allow for real-time processing (even a loss in accuracy to accomplish this might be allowable). The greatest speed increase needed is in the 3 minute gestalt calculation. Appendix G shows how the 3 minute process can be reduced to 5 seconds. After implementing Appendix G, an example of multiple image processing might be as follows:

1. llambert	1. srogers	1. bgeorge	1. mmayo
2. mkabrisky	2. mmayo	2. srogers	2. srogers
3. srogers	3. efretheim	3. llambert	3. efretheim
4. bgeorge	4. mkabrisky	4. mmayo	4. llambert
5. mmayo	5. llambert	5. efretheim	5. bgeorge

Table 6-1. Output Lists for 4 Images of a Subject

Average positions of candidates for all four images, lowest number wins first place in list (give them a number 6 if not in top 5 of a list).

efretheim	= (6+3+5+3)/4	= 4.25
srogers	= (3+1+2+2)/4	= 2.0
mkabrisky	= (2+4+6+6)/4	= 4.5
mmayo	= (5+2+4+1)/4	= 3.0
llambert	= (1+5+3+4)/4	= 3.25
bgeorge	= (4+6+1+5)/4	= 4.0

Table 6-2. Calculation of Average Position

The top candidate in this case is "srogers" because this name came up closest to the top of the candidate list on average. The output list would be as follows:

1. srogers
2. mmayo
3. llambert
4. bgeorge
5. efretheim

Note that this technique would not require a candidate to ever place in the top position, and would probably help to increase the distance between false faces and real faces.

Appendix A

Equipment List

The following equipment is used in this thesis:

1. Data General Eclipse S/250 Computer System
2. Data General Nova 2 Computer System
3. Octek 2000 Video Processing Board
4. Dage 650 Video Camera
5. Panasonic WV-5490 Monochrome Monitor
6. Tektronix 4632 Video Hard Copy Unit
7. Micro-VAX Computer System
8. DeAnza Systems Color Image Display System
9. Imaging Technology Series 100 Image Processing Board

This equipment, with documentation, is available in the AFIT Signal Processing Lab.

Appendix B

Software Listings

All programs listed above can be run independently of FACE.C. In addition, FACE.C contains copies of the others as needed so that FACE.C is the complete AFRM. FACE.C must be linked to the ITEX library as shown in the user's manual.

```
/**********************************************************************
*       Name: SUB_DEMO.C   Demo of real time subtraction techniques.  *
*                                                                     *
*       Author: Laurence C. Lambert - 1987                            *
**********************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"

/**********************************************************************/
main()
{
    unsigned    base = 0x1600;
    long                mem = 0x200000L;
    int         flag = 1, block = 8;
sethdw(base, mem, flag, block);
initialize();
subdemo(1);
subdemo(2);
}
/**********************************************************************/
#define A0 (short int)a0(i)/* These transformations are used in the  */
#define a0(i) (i & 0x003f) /* feedback lut for real time subtraction */
#define A1 (short int)a1(i)/* demo. This software was created using  */
#define a1(i) ((i & 0x0fc0) >> 6)  /* "Toolbox" (see FG-100 user's   */
#define D0(i) { data &= 0xffc0; data |= (i & 0x003f); } /* manual.   */
#define D1(i) { data &= 0xf03f; data |= ((i << 6) & 0x0fc0); } /**** */
#define INPUT 0x6000
#define abs(i) (((i) < 0) ? (-(i)) : (i))

xform1(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A1);
D0(abs(A1 - A0));
return((unsigned)data);
}

xform2(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A0);
D0(abs(A1 - A0));
return((unsigned)data);
}
```

```
/***********************************************************************/
subdemo(version)      /* demo. of real time subtraction capabilities   */
 int version;         /* of the itex system. One of these algorithms   */
{                     /* may be used in the AFRM (see demo menu)       */
 register unsigned i;/*************************************************/
rtsubtract(0);
setlut(0,0);
setinmux(6);
if (version == 1) {
 printf("\n\n\n Subtracting images as follows:");
 printf("\n\n For Image = 1 to n, Display = 2-1,3-2,4-3....n-(n-1)\n");
 printf("\n This is useful for detecting motion, as anything moving");
 printf("\n will be slightly shifted from one frame to the next.");
 printf("\n When the target stops, it disappears from view.\n");
 grab(1);
 swap6();
 setinmux(6);
 for (i=0;i<0x1000;i++) write_lut(INPUT,i,xform2(i,read_lut(INPUT,i)));
}
else {
 printf("\n\n\nPrepare background image (image #1) and press RETURN.");
 grab(0);
 getchar();
 stopgrab(1);
 swap6();
 printf("\n\n\n Subtracting images as follows:");
 printf("\n\n For Image = 1 to n, Display = 2-1,3-1,4-1,....n-1\n");
 printf("\n This is used to display the brightness diff between two");
 printf("\n scenes. The resulting ghost shows where the images vary.");
 printf("\n Since 1 scene never changes, anything that is diff in");
 printf("\n the other will show up whether it is moving or not.");
 printf("\n A target cannot hide by standing still.\n");
 setinmux(6);
 for (i=0;i<0x1000;i++) write_lut(INPUT,i,xform1(i,read_lut(INPUT,i)));
}
grab(0);
printf("\n\n\n\n\n\n Press RETURN to continue.");
getchar();
stopgrab(1);
initialize();
sclear(0);
return;
}
/***********************************************************************/
```

```
/*********************************************************************
*       Name    MTI.C    Demo of moving target detector              *
*                                                                    *
*       Author: Laurence C. Lambert - 1987                           *
*********************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"
static int j,sx,sy,fx,fy;
/*********************************************************************/
main()
{
    unsigned    base = 0x1600;
    long                mem = 0x200000L;
    int         flag = 1, block = 8;
sethdw(base, mem, flag, block);
initialize();
rtsubtract(0);
setlut(0,0);
setinmux(6);
for (j=0;j<0x1000;j++) write_lut(INPUT,j,xform2(j,read_lut(INPUT,j)));
printf(" looking for target.");
snap(1);
snap(1);
while((isolate(8,6,32)) != 1) snap(1);
printf("\n found target, acquiring 8 bit image.");
initialize();
snap(1);
}
/*********************************************************************/
int isolate(thresh,mode,size)
    int thresh; /* threshold for detection of target */
    int mode;   /* 6 bit or 8 bit image */
    int size;   /* determines min. size of target and affects speed. */
{            /* size is either 16 or 32 pixels. */
    int x,y,z;
sx = -1;
sy = -1;
fx = -1;
fy = -1;   /* Find top of target     **********************************/
for (y=size-1; y <= 255; y=y+size){/* This subroutine finds location */
 for (x = 0; x < 511; x=x+size){   /* of a moving object. If there is*/
  z = brpixel(x,y);                /* no moving object, or it is too */
  if (mode == 6) z = z & 63;       /* small then (0) is returned. If */
  if (z >= thresh) {               /* an object is found then sx,sy, */
   sy = y-(size-1);                /* fx,fy are set and (1) is re-   */
   x = 512;                        /* turned. This is done so that   */
   y = 512;                        /* all future work done on a scene*/
}}}                                /* is done on a greatly reduced   */
```

```
if (sy == -1) return(0);            /* area of the scene and hence is */
for (y=256-size; y>(sy+size-1); y=y-size){/* done faster. Thresh is  */
 for (x = 0; x <= 511; x=x+size){  /* high enough value to eliminate */
  z = brpixel(x,y);                 /* video noise but low enough to  */
  if (mode == 6) z = z & 63;        /* find small brightness differen-*/
  if (z >= thresh){                 /* ces that may occur between a   */
   fy = y + size-1; /* Find bottom. * moving object and its bkgnd.    */
   x = 512;                         /**********************************/
   y = -1;
}}}
if (fy < (sy + size)) return(0);
for (x=size-1; x <= 511; x=x+size){  /* find left side */
 for (y = 0; y < 255; y=y+size){
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   sx = x - (size-1);
   x = 512;
   y = 512;
}}}
if (sx == -1) return(0);
for (x = 512-size; x > (sx + size-1); x = x - size){
 for (y = 0; y < 255; y = y + size){  /* find right side */
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   fx = x + size-1;
   x = -1;
   y = 512;
}}}
if (fx < (sx + size)) return(0);
return(1);
}
/**********************************************************************/
#define A0 (short int)a0(i)/*These are the transforms used in the    */
#define a0(i) (i & 0x003f) /*feedback lut for the real time subtract */
#define A1 (short int)a1(i)/*demo. This software was created by using*/
#define a1(i) ((i & 0x0fc0) >> 6)  /* "toolbox" (see FG-100 manual   */
#define D0(i) { data &= 0xffc0; data |= (i & 0x003f); }  /* chapt 7) */
#define D1(i) { data &= 0xf03f; data |= ((i << 6) & 0x0fc0); } /******/
#define INPUT 0x6000
#define abs(i) (((i) < 0) ? (-(i)) : (i))

xform2(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A0);
D0(abs(A1 - A0));
return((unsigned)data);
}
/**********************************************************************/
```

```
/***********************************************************************
*       BRIGHT.C : Brightness normalization algorithm                 *
*                     will process whatever is on monitor.            *
*     Author : Laurence C. Lambert - 1987                           *
***********************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"
struct array{
       int data[512];
};
static struct array pic[512],norm[512];
static int col[512];
/**********************************************************************/
main()
{
    unsigned    base = 0x1600;
    long        mem = 0x200000L;
    int         flag = 1,block = 8;
    int         pix,avg,diff,neigh,x,y,i,j;
sethdw(base,mem,flag,block);
printf(" takes about 15 seconds to process. please wait...");
for (y=0; y<480; y++) {  /* read from video memory */
 rhline(0,y,512,pic[y].data);
}
y = 0;
for (i=0; i<512; i++) {
 col[i] = 0;              /*  setup all columns for first y value */
 for (j=y; j<y+9; j++) col[i] += pic[j].data[i];
}
for (y=1; y<471; y++) {
 for (i=0; i<512; i++) {     /* now all columns calculated faster  */
  col[i] += (pic[y+8].data[i] - pic[y-1].data[i]);
 }
 x = neigh = 0;                          /* setup first neighborhood */
 for (i=x; i<x+9; i++) neigh += col[i];
 for (x=1; x<503; x++) { /* now all other neigh are calc faster    */
  neigh += (col[x+8] - col[x-1]);

  avg = neigh/81;      /* these four lines are the heart of it all */
  pix = pic[y+4].data[x+4];        /* neighborhood size = 9x9      */
  diff = pix - avg;                /* center size = 1              */
  pix = 128 + diff;
                                         /* for awesome effects try: */
  if (pix < 0) pix = 0;                  /*  other sizes,            */
  if (pix > 255) pix = 255;              /*  pix=128+multiplier*diff,*/
  norm[y+4].data[x+4] = pix;             /*  thresholding result,    */
}}                                       /*  etc...                  */
for (y=0; y<480; y++) {
 whline(0,y,512,norm[y].data);
}
}
/**********************************************************************/
```

```
/************************************************************************
*       Name    GRAPH.C                                                 *
*                                                                       *
*       Author: Laurence C. Lambert - 1987                              *
************************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"
/***********************************************************************/
main()
{
 int i,m,x,y,z,numline,old,new;
    unsigned    base = 0x1600;
    long                mem = 0x200000L;
    int         flag = 1, block = 8;
sethdw(base, mem, flag, block);
carea(0,0,512,255,0,256,512,255);
numline = 256;
printf("\n\n\n which line to graph?(0-255)");
scanf("%d",&numline);
while (numline > -1){
 printf("\n Enter -1 if Okay, or enter new line to graph(0-255).>");
 line(0,numline,511,numline,255);
 y = numline;
 scanf("%d",&numline);
}
aclear(0,0,512,255,175);
printf("\n\n Plotting selected line.");
y = y + 256;
old = 0;
for (x=0; x<510; x++) {
 new = brpixel(x,y);
 if (new < old) {
  for (m=new; m<old+1; m++) {
   bwpixel(x,256-m,0);
 }}
 else {
  for (m=old; m<new+1; m++) {
   bwpixel(x,256-m,0);
 }}
 old = new;
}
line(0,y,511,y,0);
}
```

```
/**********************************************************************
*       Name    FACE_SIG.C face finder that looks for characteristic  *
*                          brightness variations (signature) in a thin*
*                          strip of data from the input scene.         *
*       Author: Laurence C. Lambert - 1987                             *
*       Based upon technique used on the Eclipse/Nova by E. Smith      *
**********************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"

int px[8],py[8],pz[8];

static int     sx,sy,fx,fy;
/**********************************************************************/
main()
{
    unsigned    base = 0x1600;
    long        mem = 0x200000L;
    int         flag = 1, block = 8;
sethdw(base, mem, flag, block);
          sx = 0;
          sy = 0;
          fx = 511;
          fy = 255;
          carea(0,0,511,256,0,256,511,256);
          printf("\n\n Smoothing image.");
          blur(0,0,512,256,3);
          if (findface() != 1) printf(" Face not found.");
}
/**********************************************************************/
int findface()/*looks for face in image. returns 1 if found. 0 if not*/
{                /* masks face with ellipse, and updates sx,sy,fx,fy.*/
 int x,y,z,j;    /*****************************************************/
 int size,bright,prevz,point,direc,prevdirec,nz;
 int lislope,rislope,loslope,roslope,test,radius,center;
printf("\n\n looking for face...");
for (y=sy; y<(fy+1); y++) {
 direc = 1;
 test = 0;
 bwpixel(0,y,250);/* gives indication of where facefinder is working */
 z = brpixel(sx,y);
 for (x=sx; x<fx; x=x+2) {
  prevz = z;
  prevdirec = direc;
  z = brpixel(x,y);
  if (z > prevz) direc = 1;
  if (z < prevz) direc = -1;
  if (z == prevz && direc < 0) direc = direc - 1;
  if (z == prevz && direc > 0) direc = direc + 1;
  point = test;
```

```
switch(point) {
 case 0:if (direc < 0 && prevdirec > 0) { /* possible point 1 found*/
         test = 1;
         px[1] = x - 2;
         py[1] = y;
         pz[1] = brpixel(px[1],py[1]);
        }
        break;
 case 1:if (direc > 0 && prevdirec <0) { /* possible point 2 found */
         test = 2;
         px[2] = x + (prevdirec - 1);     /* use center of plateau */
         py[2] = y;                       /* if on a plateau.      */
         pz[2] = brpixel(px[2],py[2]);
         bright = pz[1] - pz[2];
         if (bright < 10) {
          test = 0; /* not enough distance between pts 1 and 2.    */
          x = px[2];
        }}
        break;
 case 2:if (direc < 0 && prevdirec > 0) { /* possible point 3 found*/
         test = 3;
         px[3] = x - (prevdirec + 1);     /* use center of plateau */
         py[3] = y;                       /* if on a plateau       */
         pz[3] = brpixel(px[3],py[3]);
        }
        break;
 case 3:if (direc > 0 && prevdirec < 0) { /* possible point 4 found*/
         test = 4;
         px[4] = x + (prevdirec - 1);  /* use center of plateau if */
         py[4] = y;                    /* on a plateau.            */
         pz[4] = brpixel(px[4],py[4]);
         if (pz[3]-pz[4] < bright/3 || abs(pz[2]-pz[4]) > bright/4) {
          test = 0;  /* not enough diff between points 3 and 4 or  */
          x = px[2]; /* too much variation between points 2 and 4. */
        }}
        break;
 case 4:if (direc < 0 && prevdirec > 0) {/*possible point 5 found. */
         test = 5;
         px[5] = x - 2*prevdirec;
         py[5] = y;
         pz[5] = brpixel(px[5],py[5]);
         if (abs(pz[1]-pz[5]) > (bright/2)) test = 0;/*too much 1-5*/
         lislope = (pz[3] - pz[2])/(px[3] - px[2]); /* test slopes */
         rislope = (pz[3] - pz[4])/(px[4] - px[3]);
         loslope = (pz[1] - pz[2])/(px[2] - px[1]);
         roslope = (pz[5] - pz[4])/(px[5] - px[4]);
         /* check var of inner,outer,lft out-in, rght out-in slopes*/
         if (lislope>(1.4*rislope) || rislope>(1.4*lislope))test = 0;
         if (loslope>(1.4*roslope) || roslope>(1.4*loslope))test = 0;
         if (loslope > (3*lislope) || lislope > (3*loslope))test = 0;
         if (roslope > (3*rislope) || rislope > (3*roslope))test = 0;
```

```
        /* test physical distance ratios between points           */
        if ((px[4]-px[3]) > (1.3*(px[3]-px[2]))) test = 0;
        if ((px[3]-px[2]) > (1.3*(px[4]-px[3]))) test = 0;
        if ((px[2]-px[1]) > (1.5*(px[5]-px[4]))) test = 0;
        if ((px[5]-px[4]) > (1.5*(px[2]-px[1]))) test = 0;
        if ((px[5]-px[1]) < 32) test = 0; /* face too small */
        if ((px[5]-px[1]) > 150) test = 0; /* face too large */
        if (test == 0) {
         x = px[2];
         direc = 1;
       }}
       break;
case 5:size = 1.4*(px[5] - px[1]); /* approx scale of whole face  */
       /* test to see if whole face on screen */
       if ((px[3]-(size/2)) < 0 || (px[3]+(size/2)) > 511) test = 0;
       if ((py[3]-(size/2)) < 0 || (py[3]+size) > 256) test = 0;
       if (test == 0) {    /* whole face not on screen */
        x = px[2];
        direc = 1;
        break;
       }
       direc = 1;/* okay, look for nose mouth signature. */
       nz = pz[3];
       for (j=(py[3]+1); j<(py[3]+size); j++) {
        prevdirec = direc;
        prevz = nz;
        nz = brpixel(px[3],j);
        if (nz > prevz) direc = 1;
        if (nz < prevz) direc = -1;
        if (nz == prevz && direc < 0) direc = direc - 1;
        if (nz == prevz && direc > 0) direc = direc + 1;
        if (direc < 0 && prevdirec > 0) { /*possible point 6 found*/
         test = 6;
         px[6] = px[3];
         py[6] = j-1;
         pz[6] = brpixel(px[6],py[6]);
         if (py[6]-py[3]<(px[3]-px[2])/2||py[6]-py[3]>px[4]-px[2]) {
          test = 0;
          x = px[2];/*point 6 physically too close to point 3 or  */
          direc = 1;/*point 6 physically too far down.            */
          break;
         }
         j = 512;
       }}
       if (test == 6) { /*look for point 7 */
        direc = 1;
        nz = pz[6];
        for (j=(py[6]+1); j<(py[3]+size); j++) {
         prevdirec = direc;
         prevz = nz;
         nz = brpixel(px[3],j);
         if (nz > prevz) direc = 1;
         if (nz < prevz) direc = -1;
         if (nz == prevz && direc < 0) direc = direc - 1;
         if (nz == prevz && direc > 0) direc = direc + 1;
```

```
        if (direc > 0 && prevdirec < 0) { /* possible pt. 7 found*/
         test = 7;
         px[7] = px[3];
         py[7] = j-1;
         pz[7] = brpixel(px[7],py[7]);
         if ((pz[6]-pz[7]) < bright/2) {
       test = 0;  /* not enough bright diff between points 6,7    */
          x = px[2];
          direc = 1;
          break;
         }
         j = 512;
      }}}
      if (test == 7) {
       j = py[2];
       while (brpixel(px[2],j+2) < brpixel(px[2],j)) j = j + 2;
       while (brpixel(px[2],j-2) < brpixel(px[2],j)) j = j - 2;
       py[2] = j; /* the correct vertical center of the eyes */
       center = py[2];
       sx = (px[1]+(px[3]-size/2))/2; /* reset edges of face. */
       fx = (px[5]+(px[3]+size/2))/2;
       sy = center - 9*size/10;
       fy = center + 9*size/10;
       radius = size;
       circle(px[3],center+256,radius,1,2,255);
       for (j=px[1]; j<px[5]; j++) bwpixel(j,py[1],0);
       for (j=py[3]; j<py[7]; j++) bwpixel(px[3],j,0);
       for (j=1; j<8; j++) bwpixel(px[j],py[j],250);
       rectangle(sx,sy+256,fx-sx,fy-sy,255);
       fill(sx+1,sy+257,255,255);
       fill(fx-1,sy+257,255,255);
       fill(sx+1,fy+255,255,255);
       fill(fx-1,fy+255,255,255);
       return(1);
      }
      if (test != 7) {   /* point 7 was never found. */
       test = 0;
       x = px[2];
       direc = 1;
      }
      break;
}}}
return(0);
}
/***********************************************************************/
```

```
/**********************************************************************
*     Name      FACE - AUTONOMOUS FACE RECOGNITION MACHINE             *
*                                                                      *
*     Author: Laurence C. Lambert - 1987                              *
*     Based on the Data General (Eclipse/Nova) AFRM by E. Smith       *
**********************************************************************/



#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"
#include <math>
static int option,test,sy,sx,fy,fx,nf,x,y,z;
int i,k;          /* i = size of tlist, k = size of ilist */
struct list{
       char name[10];
       int  num;
       int  win1x,win1y,win2x,win2y,win3x,win3y;
       int  win4x,win4y,win5x,win5y,win6x,win6y;
};
static struct list tlist[400] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};
static struct list ilist[100] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};
static double gauss[257];



/***********************************************************************/
main()                                                         /* 23 */
{
    unsigned     base = 0x1600;
    long         mem = 0x200000L;
    int          flag = 1,block = 8;
sethdw(base,mem,flag,block);
initialize();
sclear(100,1);
cls();
printf("\n            Initializing hardware and reading dbase files.");
printf("\n            Please turn on the video monitor and the camera.");
text(120,200,0,8,0,"AFRM");
text(110,415,0,1,0," AIR FORCE INSTITUTE OF TECHNOLOGY");
text(110,430,0,1,0,"    SIGNAL PROCESSING LABORATORY");
text(110,445,0,1,0,"AUTONOMOUS FACE RECOGNITION MACHINE");
text(110,460,0,1,0,"                1987");
del();
i = readfile("[face.dbase]train.dat;1",tlist);
k = readfile("[face.dbase]others.dat;1",ilist);
rtransa();            /* setup gaussian xform for later use in gestalt() */
nf = sx = sy = 0;
fx = fy = 511;
menu1();
}
```

```
/***************************************************************************/
menu1()
{
    char t2[30],t3[30],temp[10],ch[2]; /* j,l and m are counters */
    int ver[5],j,l,m;  /* ver[1-4] = file version numbers to train on.*/
    int r;
    int rot[5] = {-20,-10,0,10,20};
for(;;) {
 cls();
 printf("\n                        AUTONOMOUS FACE RECOGNITION MACHINE\n ");
 printf("\n                               ***** MAIN MENU  *****\n ");
 printf("\n                            1:ACQUIRE IMAGES");
 printf("\n                            2:FIND FACES");
 printf("\n                            3:GESTALT AND IDENTIFY / SAVE\n");
 printf("\n                            4:DISPLAY CONTENTS OF DATABASE");
 printf("\n                            5:DELETE A SUBJECT");
 printf("\n                            6:DELETE AN IMAGE");
 printf("\n                            7:TRAIN\n");
 printf("\n                            8:DEMONSTRATION\n");
 printf("\n                            0:QUIT\n\n\n\n\n\n>");
 scanf("%d",&option);
 cls();
 switch(option){
  case 0:
   printf("\n\n saving DBASE files...");
   writefile("[face.dbase]train.dat;1",tlist,i);
   writefile("[face.dbase]others.dat;1",ilist,k);
   cls();
   printf("        Please turn off the video monitor and the camera.");
   printf("\n\n\n\n\n\n\n\n\n\n\n\n");
   return;
  case 1:
   menu2();
   break;
  case 2:
   printf("\n Sharpen image? (Y/N) >");
   scanf("%s",ch),
   if (ch[0] == 'y' || ch[0] == 'Y') {
     sharpen(sx,sy,fx-sx,fy-sy,3);
   }
   facemap();
   break;
  case 3:
   facerec(1);
   break;
  case 4:
   display(tlist,i,8);
   display(ilist,k,5);
   prtc();
   break;
```

```
case 5:
 display(tlist,i,8);
 printf("\n\n                         ***** DELETE SUBJECT *****");
 printf("\n\n Are you sure (Y/N)? >");
 scanf("%s",temp);
 if (temp[0] != 'Y' && temp[0] != 'y') break;
 printf("\n\n Do you wish to save ");
 printf("subject's training pictures as .IMG files (Y/N)? >");
 test = 0;
 scanf("%s",temp);
 if (temp[0] == 'Y' || temp[0] == 'y') test = 2;
 printf("\n\n Enter subject's name. >");
 scanf("%s",temp);
 printf("\n");
 for (j=1; j<(i+1); j=j+4) {
  if ((strcmp(tlist[j].name,temp)) == 0) {
   if (test == 2) {
    l = 0; /* look for highest existing version of .IMG file. */
    for (m=1; m<(k+1); m++)
     if (strcmp(ilist[m].name,temp)==0 && ilist[m].num>>l) l=ilist[m].num;
    l = l + 1; /* add 1 to highest version to get new version. */
    for (m=1; m<5; m++) { /* put 4 new versions into ilist. */
     k = k + 1;
     ilist[k].name[0] = '\0';
     strcat(ilist[k].name,temp);
     ilist[k].num = l;
     ilist[k].win1x = tlist[j+(m-1)].win1x;
     ilist[k].win1y = tlist[j+(m-1)].win1y;
     ilist[k].win2x = tlist[j+(m-1)].win2x;
     ilist[k].win2y = tlist[j+(m-1)].win2y;
     ilist[k].win3x = tlist[j+(m-1)].win3x;
     ilist[k].win3y = tlist[j+(m-1)].win3y;
     ilist[k].win4x = tlist[j+(m-1)].win4x;
     ilist[k].win4y = tlist[j+(m-1)].win4y;
     ilist[k].win5x = tlist[j+(m-1)].win5x;
     ilist[k].win5y = tlist[j+(m-1)].win5y;
     ilist[k].win6x = tlist[j+(m-1)].win6x;
     ilist[k].win6y = tlist[j+(m-1)].win6y;
     l++;
    }
    l = l - 4;
    for (m=0; m<4; m++) { /* now create .IMG and .PIC file names, */
     t2[0] = '\0';
     t3[0] = '\0';
     strcat(t2,"[face.dbase]\0");
     strcat(t2,tlist[j].name);
     strcat(t3,t2);
     strcat(t2,".pic;\0");
     strcat(t3,".img;\0");
     ch[1] = '\0';
     ch[0] = tlist[j+m].num + '0'; /* convert int to char */
     strcat(t2,ch);
     ch[0] = l + '0';
     strcat(t3,ch);
     copyfile(t2,t3); /* copy .PIC > .IMG */
```

```
          l = l + 1;
        }}
        for (m=j; m<(i-3); m++) tlist[m] = tlist[m+4]; /*del from tlist*/
        printf("\n");
        i = i - 4;
        for (m=1; m<5; m++) { /* delete .PIC files. */
         t2[0] = '\0';
         strcat(t2,"[face.dbase]\0");
         strcat(t2,temp);
         strcat(t2,".pic;\0");
         t3[0] = m + '0';
         t3[1] = '\0';
         strcat(t2,t3);
         printf("%s%s","\n Deleting ",t2);
         delete(t2);
        }
        j = i + 2; /* forces end of loop through tlist */
        test = 1; /* indicates that subject was found */
      }}
      if (test != 1) printf("\n\n Subject not found.");
      else printf("\n\n Subject deleted.");
      prtc();
      break;
     case 6:
      display(ilist,k,5);
      printf("\n\n                    ***** DELETE IMAGE *****");
      printf("\n\n Are you sure (Y/N)? >");
      scanf("%s",temp);
      if (temp[0] != 'Y' && temp[0] != 'y') break;
      printf("\n\n Enter subject's name. >");
      scanf("%s",temp);
      printf("\n\n Enter version number. >");
      scanf("%s",ch);
      printf("\n");
      test = 0;
      for (j=1; j<(k+1); j++) {
       if (strcmp(ilist[j].name,temp) == 0) {
        if (ilist[j].num == (ch[0] - '0')) {
         t2[0] = '\0';
         strcat(t2,"[face.dbase]\0");
         strcat(t2,ilist[j].name);
         strcat(t2,".img;\0");
         strcat(t2,ch);
         printf("%s%s","\n Deleting ",t2);
         delete(t2);
         for (m=j; m<k; m++) ilist[m] = ilist[m+1];
         k = k - 1;
         j = k + 2;
         test = 1;
      }}}
      if (test != 1) printf("\n\n Image file not found.");
      else printf("\n\n Image file deleted.");
      prtc();
      break;
```

```
case 7:
 display(tlist,i,8);
 display(ilist,k,5);
 printf("\n\n                                    ***** TRAIN *****");
 printf("\n\n Enter person's name. >");
 scanf("%s",temp);
 test = 0;
 for (l=1; l<(i+1); l=l+4)                    /* test name */
  if ((strcmp(tlist[l].name,temp)) == 0) test = 1;
 if (test == 1) {
  printf("\n That name already exists in the training file.");
  prtc();
  break;
 }
 for (l=1; l<(k+1); l++)
  if ((strcmp(ilist[l].name,temp)) == 0) test = 2;
 if (test != 2) {
  printf("\n There are no image files with that name.");
  prtc();
  break;
 }
 printf("\n\n You must enter 4 valid (and unique)");
 printf(" file version numbers.");
 printf("\n (Enter -1 to quit training procedure)");
 for (j=1; j<5; j++) {
  printf("\n Enter version number for training file # ");
  printf("%d%s",j,">");
  scanf("%s",ch);
  ver[j] = ch[0] - '0';
  if (ver[j] > 0) {
   test = 0;
   for (l=1; l<j; l++) {
    if (ver[l] == ver[j]) {
     display(ilist,k,5);
     printf("\n\n You already selected:");
     for (m=1; m<j; m++) printf("%s%d"," ",ver[m]);
     j = j - 1;
     test = 1;
   }}
   if (test != 1) {
    for (l=1; l<(k+1); l++) {
     if (strcmp(ilist[l].name,temp) == 0 && ilist[l].num == ver[j]) {
      test = 1;
      l = k + 2;
    }}
```

```
        if (test == 0) {
         display(ilist,k,5);
         printf("%s%d","\n\n File version #",ver[j]);
         printf(" not found, try another.");
         if (j != 1) {
          printf(" (You already selected:");
          for (l=1; l<j; l++) printf("%s%d"," ",ver[l]);
          printf(")");
         }
         j = j - 1;
     }}}
     else j = 5;
    }
    if (j == 6) break;
    for (j=1; j<5; j++) {
     i = i + 1;
     for (l=1; l<(k+1); l++) {
      if (strcmp(ilist[l].name,temp) == 0 && ilist[l].num == ver[j]) {
       tlist[i] = ilist[l];      /*find proper gestalt file in ilist,*/
       tlist[i].num = j;         /*put in tlist, */
       for (m=1; m<k; m++) ilist[m] = ilist[m+1];
       k = k - 1;                /*delete from ilist,  */
     }}
     t2[0] = '\0';               /*create .PIC and .IMG file names, */
     t3[0] = '\0';
     strcat(t2,"[face.dbase]\0");
     strcat(t2,temp);
     strcat(t3,t2);
     strcat(t2,".pic;\0");
     strcat(t3,".img;\0");
     ch[1] = '\0';
     ch[0] = ver[j] + '0';
     strcat(t3,ch);
     ch[0] = j + '0';
     strcat(t2,ch);
     printf("\n");
     copyfile(t3,t2);         /* copy .IMG > .PIC */
     printf("%s%s","\n Deleting ",t3);
     delete(t3);                /*and finally, delete the .IMG files */
    }
    printf("%s%s%s","\n\n The training file now contains <",temp,">.");
    prtc();
    break;
   case 8:
    menu3();
    break;
   default:
    break;
}}
}
```

```
/***************************************************************************/
menu2()
{
    int cam;
    char name[50],t1[50],temp[2];
for (;;) {
 cls();
 printf("\n                    ***** ACQUISITION OF IMAGES *****\n");
 printf("\n                        0:RETURN TO MAIN MENU\n");
 printf("\n                        1:STATIONARY TARGET");
 printf("\n                        2:MOVING TARGET");
 printf("\n                        3:LOAD IMAGE FROM MEMORY\n");
 printf("\n                        4:SAVE IMAGE IN [FACE]\n");
 printf("\n                        5:SET CAMERA PORT");
 printf("\n                        6:CAMERA CHECK");
 printf("\n                        7:RE-INITIALIZE HARDWARE\n\n\n\n\n\n>");
 scanf("%d",&option);
 cls();
 switch (option) {
  case 0:
   return;
  case 1:
   nf = sx = sy = 0;
   fx = fy = 511;
   getchar();
   printf("\n                  ***** STATIONARY TARGET *****");
   printf("\n\n Acquire new image (Y/N)? >");
   scanf("%s",temp);
   if (temp[0] == 'Y' || temp[0] == 'y') {
    grab(0);
    prtc();
    stopgrab(1);
    }
   break;
  case 2:
   nf = 0; /* this algorithm sets sx,sy,fx,fy to target's location */
   printf("\n                  ***** MOVING TARGET *****\n\n");
   getchar();
   printf("\n Prepare background image and press RETURN to continue. >");
   waitvb();             /* The aclear() is used in this routine to */
   grab(0);              /* clear the 1st 16 columns of the image   */
   getchar();            /* because of an X_SPIN delay of the image.*/
   stopgrab(1);          /* Therefore the 256x512 image is really   */
   setreg(X_SPIN,0);     /* only a 256x496 image.                   */
   snap(1);
   aclear(0,0,16,768,0);
   setreg(SCROLL,256);
   printf("\n\n Prepare subject image and press RETURN to continue. >");
   waitvb();        /* Scrolling 256 stores the background image     */
   grab(0);         /* off the screen area.  Scroll 0 brings it      */
   getchar();       /* back. I have used the setreg function instead */
   stopgrab(1);     /* of scroll because of the problem with defini- */
   setreg(X_SPIN,0); /* tions in the library. (see the comment       */
   snap(1);         /* obtained when linking this program).          */
```

```
        aclear(0,0,16,512,0);
        setreg(SCROLL,0);
        printf("\n\n Subtracting images and locating target.");
        oparea(0,256,512,255,0,0,512,255,5,1);
        if (isolate(20,8,16) == 0) {  /* Isolate target from surroundings */
         printf("\n Could not find target. Press RETURN to continue. >");
         getchar();
         sx = sy = 0;
         fx = fy = 511;
         break;
        }
        carea(sx+1,sy+257,fx-sx-1,fy-sy-1,sx+1,sy+1,fx-sx-1,fy-sy-1);
        aclear(0,255,512,256,0);
        break;
       case 3:
        printf("\n                      ***** LOAD IMAGE FROM MEMORY *****");
        printf("\n\n\n Enter complete file specification.");
        printf("\n ([directory.subdirectory]filename.ext;version)\n\n >");
        scanf("%s", name);
        printf("\n\n\n Loading file...");
        if (readim(0,0,511,480,name,"nocomm") == -1) {
         printf("\n\n\n File not found.");
         prtc();
        }
        nf = sx = sy = 0;
        fx = fy = 511;
        break;
       case 4:
        t1[0] = '\0';
        strcat(t1,"[face]\0");
        printf("\n Will save ******ENTIRE****** screen as 8-bit image in ");
        printf("directory [FACE].\n\n\n Enter name (including EXT)\n\n >");
        scanf("%s", name);        /* I want to make sure that the DBASE    */
        strcat(t1, name);         /* directory is not touched by this save.*/
        printf("\n\n\n\n Saving image...");  /* Hence the directory name is*/
        saveim(0,0,511,480,0,t1,"nocomm");    /* not allowed to vary.      */
        break;
       case 5:
        printf("\n Select camera port (0,1 or 2) >");
        scanf("%d",&cam);
        setcamera(cam);
        break;
       case 6:
        printf("\n\n                      ***** CAMERA CHECK *****\n\n");
        grab(0);
        prtc();
        stopgrab(1);
        nf = sx = sy = 0;
        fx = fy = 511;
        break;
```

```
    case 7:
     initialize();
     nf = sx = sy = 0;
     fx = fy = 511;   /*************************************************/
     sclear(100);     /* using 100 gives a clean screen that is not too */
     break;           /* dark to tell whether the monitor is on/off     */
    default:          /* This is to prevent accidently leaving it on.   */
     break;           /*************************************************/
  }}
  }
/*****************************************************************************/
menu3()                                                              /* 402 */
{
    char temp[10],ch[2],t3[30];
    int version,j;
for (;;) {
 cls();
 printf("\n                        ***** DEMONSTRATION *****\n");
 printf("\n                        0:RETURN TO MAIN MENU\n");
 printf("\n                        1:IDENTIFY A PERSON");
 printf("\n                        2:TOTAL SYSTEM\n\n\n\n\n\n\n>");
 scanf("%d",&option);
 switch (option) {
  case 0:
   return;
  case 1:
   display(ilist,k,5);
   printf("\n\n                     ***** IDENTIFY A PERSON *****");
   printf("\n\n Enter person's name. >");
   scanf("%s",temp);
   printf("\n\n Enter version number. >");
   scanf("%s",ch);
   printf("\n");
   test = -1;
   for (j=1; j<(k+1); j++) {
    if (strcmp(ilist[j].name,temp) == 0) {
     if (ilist[j].num == (ch[0] - '0')) {
      t3[0] = '\0';
      strcat(t3,"[face.dbase]\0");
      strcat(t3,ilist[j].name);
      strcat(t3,".img;\0");
      strcat(t3,ch);
      test = j;
      j = k + 2;
   }}}
   if (test == -1) printf("\n\n Image file not found.");
   else {
    nf = sx = sy = 0;
    fx = fy = 511;
    sclear(0,1);
    readim(200,30,200,200,t3,"nocomm");
    text(200,10,0,1,200,temp);
    recognize(test);
   }
   break;
```

```
  case 2:
   afrm();
   break;
  default:
   break;
}}
}



/**************************************************************************/
prtc()
{
printf("\n\n Press RETURN to continue. >");
getchar();
getchar();
return;
}



/**************************************************************************/
cls()
{
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
return;
}
/**************************************************************************/



static int pix,avg,diff,neigh,threshold,ne,nn,nm;
static int col[512];
struct image{
       int data[512];
};
struct feat{
       int sx,sy,fx,fy,xcenter,ycenter,pix,xsize,ysize,used;
};
struct whole{
       int x,y,dx,dy,leye,reye,teye,beye,tnose,cmouth;
       int center,xellipse,yellipse,radius;
};
static struct image  pic[512],norm[512],temp[512];
static struct feat   eye[100],nose[100],mouth[100];
static struct whole  face[10];
```

```
/*********************************************************************/
int facemap()
{
     int i,j,k,l;
     char name[30];
del();
cls();
printf("\n processing image...");
bright_norm();
ne = nn = nm = nf = 0;
featuremap();
for (i=1; i<ne; i++) {
 if (eye[i].used == 0) {
  for (j=1; j<ne+1; j++) {                        /* look for a matching eye */
   if (eye[j].sx > eye[i].fx && eye[j].used == 0) {          /* try eye[j] */
    if (abs(eye[j].pix - eye[i].pix) < eye[j].pix/2) {/* pix nums okay */
     if (eye[j].ycenter > eye[i].sy &&
         eye[j].ycenter < eye[i].fy) {                   /* close in height */
      if (eye[j].sx < eye[i].fx+2*eye[i].xsize) {           /* near enough */
       for (k=1; k<nn+1; k++) {                          /* look for a nose */
        if (nose[k].sy > eye[i].fy && nose[k].used == 0) {/* try nose[k] */
         if (nose[k].xcenter > eye[i].sx &&
             nose[k].xcenter < eye[j].fx) {                   /* between eyes */
          for (l=1; l<nm+1; l++) {                      /* look for a mouth */
           if (mouth[l].ycenter > nose[k].fy &&
               mouth[i].used == 0) {                           /* below nose */
            if (mouth[l].ycenter < eye[i].fy+4*eye[i].ysize) { /* near*/
             if (mouth[l].xcenter > eye[i].sx &&                  /* enough */
                 mouth[l].xcenter < eye[j].fx) {              /* between eyes */
              nf = nf+1; /* all features found and cond met for a face. */
              eye[i].used = eye[j].used = 1;
              nose[k].used = mouth[l].used = 1;
              face[nf].dx = 9*(eye[j].xcenter - eye[i].xcenter)/4;
              face[nf].dy = 2*(mouth[l].ycenter - eye[i].sy);
              face[nf].x = (eye[j].xcenter+eye[i].xcenter)/2 -
                                                       face[nf].dx/2;
              face[nf].y = mouth[l].ycenter - 4*face[nf].dy/5;
              face[nf].leye = eye[i].sx - face[nf].x;
              face[nf].reye = eye[j].fx - face[nf].x;
              face[nf].teye = (eye[i].sy + eye[j].sy)/2 - face[nf].y;
              face[nf].beye = (eye[i].fy + eye[j].fy)/2 - face[nf].y;
              face[nf].tnose = nose[k].sy - face[nf].y;
              face[nf].cmouth = mouth[l].ycenter - face[nf].y;
              face[nf].center = face[nf].dx/2;
              face[nf].xellipse = face[nf].dx/2 + face[nf].x;
              face[nf].yellipse = face[nf].dy/2 + face[nf].y;
              face[nf].radius = face[nf].dx;
              circle(face[nf].xellipse,face[nf].yellipse,
                     face[nf].radius,1,2,255);
              rectangle(eye[i].sx,eye[i].sy,eye[j].fx - eye[i].sx,
                        mouth[l].ycenter - eye[i].sy,255);
              l = k = j = 500;
}}}}}}}}}}}}}}
```

```
if (nf == 0) return(0);
printf("\n Saving brightness normalized faces to disk...");
for (y=0; y<480; y++) whline(0,y,512,norm[y].data);
name[0] = '\0';
strcat(name,"bnorm.img\0");
for (i=1; i<nf+1; i++) {
 printf("\n %s%s%d",name,";",i);
 circle(face[i].xellipse,face[i].yellipse,face[i].radius,1,2,255);
 rectangle(face[i].x,face[i].y,face[i].dx,face[i].dy,255);
 fill(face[i].x+1,face[i].y+1,255,255);
 fill(face[i].x+face[i].dx-1,face[i].y+face[i].dy-1,255,255);
 fill(face[i].x+1,face[i].y+face[i].dy-1,255,255);
 fill(face[i].x+face[i].dx-1,face[i].y+1,255,255);
 saveim(face[i].x,face[i].y,face[i].dx,face[i].dy,0,name,"nocomm");
}
printf("\n Also saving original faces...");
for (y=0; y<480; y++) whline(0,y,512,temp[y].data);
name[0] = '\0';
strcat(name,"orig.img\0");
for (i=1; i<nf+1; i++) {
 printf("\n %s%s%d",name,";",i);
 rectangle(face[i].x,face[i].y,face[i].dx,face[i].dy,255);
 saveim(face[i].x,face[i].y,face[i].dx,face[i].dy,0,name,"nocomm");
}
return(1);
}
```

```
/***********************************************************************/
featuremap()
{
 int fill,test,ymin,ymax,xmax,i,j,dy,dx,ytest,xtest,bx;
 char type;
for (y=sy+14; y<fy-14; y++) {     /* begin and end with 14 pixel margins */
 test = 0;
 for (x=sx; x<fx-14; x++) {      /* see if line is touching top of object */
  if (pic[y+1].data[x] == 0) {       /* these checks are done like this */
   if (pic[y].data[x] == 100) {      /* for speed. */
    if (pic[y].data[x-1]+pic[y].data[x+1] == 200) {
     if (pic[y].data[x-2]+pic[y].data[x+2] == 200) {
      if (pic[y].data[x-3]+pic[y].data[x+3] == 200) {
       if (pic[y].data[x-4]+pic[y].data[x+4] == 200) {
        if (pic[y].data[x-5]+pic[y].data[x+5] == 200) {
         test = 1;
         bx = x - 50;
         if (bx < 14) bx = 14;
         x = 512;
 }}}}}}}}
 if (test == 1) {             /* okay, for this line find the object(s) */
  for (x=bx; x<498; x++) {
   test = 0;
   type = 'u';
   if (pic[y].data[x] == 100) {                         /* possible corner */
    ymax = y + 40;
    if (ymax>479) ymax = 479;
    for (i=y; i<ymax+1; i++) {                   /* how far is line white? */
     if (pic[i].data[x] == 0) {
      ymax = i - 1;
      i = 512;
    }}
    if (ymax > y+1) {
     for (i=y+1; i<ymax; i++) {
      if (pic[i].data[x+1] == 0) {             /* something touching line */
       dy = i;
       test = 1;
       i = 512;
   }}}}
   if ( test == 1) {                                      /* left side ok */
    xmax = x + 50;
    if (xmax>498) xmax = 498;
    for (i=x; i<xmax+1; i++) {                   /* how far is line white? */
     if (pic[y].data[i] == 0) {
      xmax = i-1;
      i = 512;
    }}
    test = 0;
    if (xmax > x+1) {
     for (i=x+1; i<xmax; i++) {
      if (pic[y+1].data[i] == 0) {             /* something touching line */
       dx = i;
       test = 1;
       i = 512;
   }}}}
```

```
if (test == 1) {                          /* at the border of unknown object */
 test = 0;
 while (dx < xmax+1 && dy < ymax+1 && test == 0) {
  ytest = 0;
  while (dy < ymax+1 && ytest == 0) {          /* try to go across to dx */
   ytest = 1;                                         /* assume success */
   for (i=x; i<dx+1; i++)
    if (pic[dy].data[i] == 0) ytest = 0;
   if (ytest == 0) dy = dy + 1;
  }
  if (ytest == 1) {
   xtest = 0;
   while (dx<xmax+1 && xtest == 0) {             /* try to go down to dy */
    xtest = 1;                                        /* assume success */
    for (i=y; i<dy+1; i++)
     if (pic[i].data[dx] == 0) xtest = 0;                      /* failed */
    if (xtest == 0) dx = dx + 1;
   }
   if (xtest == 1) {                               /* recheck present dy */
    for (i=x; i<dx+1; i++)
     if (pic[dy].data[i] == 0) ytest = 0;                      /* failed */
    if (xtest == 1 && ytest == 1) test = 1;
}}}}
if (test == 1) {                        /* successfully blocked in object */
 if ((dy-y) > 3*(dx-x)) type = 't';                  /* too tall and thin */
 if ((dx-x) < 7) type = 't';                                 /* too small */
}
if (test == 1 && type == 'u') {
 fill = 0;
 for (j=y+1; j<dy; j++)
  for (i=x+1; i<dx; i++) if (pic[j].data[i] == 0) fill++;
 if (fill < (dx-x)*(dy-y)*3/10) test = 0;        /* less than 30% solid */
}
if (test == 1 && type == 'u') {
 if (dx-x > 2*(dy-y)) {                                /* possible mouth */
  rectangle(x,y,dx-x,dy-y,0);
  type = 'm';
  nm = nm + 1;
  mouth[nm].xcenter = (dx+x)/2;
  mouth[nm].ycenter = (dy+y)/2;
  mouth[nm].sy = y;
  mouth[nm].used = 0;
}}
```

```
if (test == 1 && type != 't') {
 fill = 0;
 ymax = dy+(2*(dy-y)/3);
 if (ymax < 480) {
  for (j=dy+1; j<ymax; j++)                       /* chk for space below */
   for (i=x; i<dx; i++) if (pic[j].data[i] == 0) fill++;
  if (fill<(ymax-dy+1)*(dx-x)*10/100) { /* less than 10% of area filled */
   type = 'e';
   ne = ne + 1;
   eye[ne].xcenter = (dx+x)/2;
   eye[ne].ycenter = (dy+y)/2;
   eye[ne].pix = (dx-x) * (dy-y);
   eye[ne].xsize = dx - x;
   eye[ne].ysize = dy - y;
   eye[ne].sx = x;
   eye[ne].fx = dx;
   eye[ne].sy = y;
   eye[ne].fy = dy;
   eye[ne].used = 0;
   rectangle(x,y,dx-x,dy-y,0);                /* (1x1 <= size <= 20x20) */
 }}
 fill = 0;
 ymin = y-(dy-y);
 if (ymin > 0) {
  for (j=ymin; j<y; j++)                        /* chk for space above */
   for (i=x; i<dx; i++) if (pic[j].data[i] == 0) fill++;
  if (fill < (y-ymin)*(dx-x)*10/100) { /* less than 10% of area filled */
   nn = nn + 1;
   nose[nn].xcenter = (dx+x)/2;
   nose[nn].ysize = dy - y;
   nose[nn].fy = dy;
   nose[nn].sy = y;
   nose[nn].pix = (dx-x) * (dy-y);
   nose[nn].used = 0;
   rectangle(x,y,dx-x,dy-y,0);                /* (1x1 <= size <= 20x20) */
}}}}}}
return;
}
```

```
/************************************************************************/
bright_norm()       /* norm will contain brightness normalized scene */
{                   /* (bright areas set to 128, dark areas= 128-diff*/
int i,j;           /* pic will contain the dark objects of the scene */
                         /* (uses variable threshold, binary output) */
sx = sx - 14;
if (sx < 0) sx = 0;
sy = sy - 14;
if (sy < 0) sy = 0;
fx = fx + 14;
if (fx > 512) fx = 512;
fy = fy + 14;
if (fy > 480) fy = 480;
for (y=0; y<480; y++) for (x=0; x<512; x++) norm[y].data[x] = 0;
for (y=0; y<480; y++) rhline(0,y,512,pic[y].data);
y = sy;
for (i=sx; i<fx; i++) {
 col[i] = 0;                        /*  setup all columns for first y value */
 for (j=y; j<y+30; j++) col[i] += pic[j].data[i];
}
for (y=sy+1; y<fy-30; y++) {         /* now all columns calculated faster  */
 for (i=sx; i<fx; i++) col[i] += (pic[y+29].data[i] - pic[y-1].data[i]);
 x = sx;
 neigh = 0;                                   /* setup first neighborhood */
 for (i=x; i<x+30; i++) neigh += col[i];
 for (x=sx+1; x<fx-30; x++) {/* now all other neigh are calculated faster */
  neigh += (col[x+29] - col[x-1]);
  avg = neigh/900;
  pix = pic[y+14].data[x+14];

  if (pix < avg) norm[y+14].data[x+14] = 128 - (avg - pix);
  else norm[y+14].data[x+14] = 128;
  if (norm[y+14].data[x+14] < 0) norm[y+14].data[x+14] = 0;

  threshold = 80*avg/100 + 7;            /* add 7 because noise is +/- 7 */
  if (pix < threshold) temp[y+14].data[x+14] = 0;               /* dark=0 */
  else temp[y+14].data[x+14] = 100;                       /* else light=100 */
}}
for (y=sy+14; y<fy-14; y++) {                              /* cleanup noise */
 for (x=sx+14; x<fx-14; x++) {
  pic[y].data[x] = temp[y].data[x];
  if (temp[y].data[x] == 0) {
   if (temp[y].data[x-1]+temp[y].data[x+1]+temp[y].data[x+2] > 0) {
    pic[y].data[x] = 100;
}}}}
for (y=sy+14; y<fy-14; y++) {
 for (x=sx+14; x<fx-14; x++) {
  if (pic[y].data[x] == 0) {
   if (pic[y].data[x-1]+pic[y-1].data[x]+pic[y+1].data[x]+
       pic[y].data[x+1] > 200) {
    pic[y].data[x] = 100;
}}}}
```

```
 for (x=fx-14; x>sx+14; x--) {
  if (pic[y].data[x] == 0) {
   if (pic[y].data[x-1]+pic[y-1].data[x]+pic[y+1].data[x]+
       pic[y].data[x+1] > 200) {
    pic[y].data[x] = 100;
}}}}
for (y=0; y<480; y++) rhline(0,y,512,temp[y].data);
for (y=0; y<480; y++) whline(0,y,512,pic[y].data);
return;
}



/***************************************************************************/
int getint(fp)          /* used by readfile() to read in the gestalt      */
 FILE *fp;              /* values that are stored in the DAT files. They  */
{                       /* are stored in columns of 5 characters. (see    */
 int i,number;          /* sample printout of .DAT file ).                */
 char c;                /**************************************************/
number = 0;
for (i=0; i<5; i++) {
 c = getc(fp);
 if (c != ' ') number = (number * 10) + (c - '0');
}
return(number);
}


/********************************************************************************/
copyfile(src,dest)
 char src[],dest[];           /* copies Itex image files. */
{
 char t9[100];

t9[0] = '\0';
strcat(t9,"copy \0");
strcat(t9,src);
strcat(t9," \0");
strcat(t9,dest);
printf("\n %s",t9);
system(t9);
return;
}
```

```
/*************************************************************************/
int readfile(name,str) /* used to read in the DAT files upon main menu */
 char name[];          /* selection = 2 (care & feeding of database).  */
 struct list str[];    /************************************************/
{
 FILE *fp,*fopen();
 int i,j,c;
fp = fopen(name,"r");
i = 0;
while ((c = getc(fp)) != '*') { /* the star denotes the EOF */
 i = i + 1;
 str[i].name[0] = c;
 for (j=1; j<10; j++) {
  c = getc(fp);
  if (c != ' ') str[i].name[j] = c;
 }
 str[i].name[j+1] = '\0';
 str[i].num = getint(fp);
 str[i].win1x = getint(fp);
 str[i].win1y = getint(fp);
 str[i].win2x = getint(fp);
 str[i].win2y = getint(fp);
 str[i].win3x = getint(fp);
 str[i].win3y = getint(fp);
 str[i].win4x = getint(fp);
 str[i].win4y = getint(fp);
 str[i].win5x = getint(fp);
 str[i].win5y = getint(fp);
 str[i].win6x = getint(fp);
 str[i].win6y = getint(fp);
 c = getc(fp);                /* read newline character */
}
fclose(fp);
return(i);
}                                                                /* 814 */
/*******************************************************************************/
writefile(name,str,i)     /* used to write updated DAT files to disk when    */
 char name[];             /* user is done modifying the database and selects */
 struct list str[];       /* menu option = 0 (Return to main menu).          */
 int i;                   /***************************************************/
{
 FILE *fp,*fopen();
 int j;
delete(name);
fp = fopen(name,"w");
for (j=1; j<(i+1); j++) {
 fprintf(fp, "%-10s%5d", str[j].name, str[j].num);
 fprintf(fp, "%5d%5d", str[j].win1x, str[j].win1y);
 fprintf(fp, "%5d%5d", str[j].win2x, str[j].win2y);
 fprintf(fp, "%5d%5d", str[j].win3x, str[j].win3y);
```

```
 fprintf(fp, "%5d%5d", str[j].win4x, str[j].win4y);
 fprintf(fp, "%5d%5d", str[j].win5x, str[j].win5y);
 fprintf(fp, "%5d%5d%s", str[j].win6x, str[j].win6y, "\n");
}
fprintf(fp, "*");
fclose(fp);
return;
}
/*********************************************************************/
display(str,k,m)      /* m = 5 or 8 depending on # columns desired     */
 struct list str[];   /* m = 5 for ilist displays, 8 for tlist displays */
 int k,m;             /* l = present column being printed on screen    */
{                     /* j counts by 1 or 4 depending on value of m    */
 int j,l;             /* this is due to format of tlist file; there are */
l = 0;                /* sets of 4 lines all with the same name and the */
if (m == 5) {         /* name only needs to be printed once.          */
 printf("\n\n The AFRM has the following .IMG files:\n");
 printf(" --------------------------------------\n");
}
else {
 printf(" The AFRM is trained on the following subjects:\n");
 printf(" ---------------------------------------------\n");
}
for (j=1; j<(k+1); j=j+(m-4)) {
 l = l + 1;
 if (l == m) {
  l = 1;
  printf("\n");
 }
 if (m == 5) printf("%11s%s%d",str[j].name, ".img;", str[j].num);
 else printf("%11s",str[j].name);
}
return;
}




/*************************************************************************/
static double cray[129][129],rinp[129];
static double jr3d3,ir3d3; /* scaled gestalt values returned from cortran16 */
static int ix,iy; /* window sizes used by cortran16 */
```

```
/*************************************************************************/
gestalt(m)       /* Values range from 0 to 128  */
int m; /* m = face number */
{
     int x,y;
line(256,0,256,512,0);
line(0,256,512,256,0);
line(384,0,384,512,0);
line(128,256,128,512,0);
/* left half: whole head */
carea(sx,sy,face[m].dx/2,face[m].dy,270,sy,face[m].dx/2,face[m].dy);
/* right half: whole head */
carea(sx+face[m].dx/2,sy,face[m].dx/2,face[m].dy,
                                         400,sy,face[m].dx/2,face[m].dy);
/* top half: top to tnose */
carea(sx,sy,face[m].dx,face[m].tnose,15,sy+256,face[m].dx,face[m].tnose);
/* internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].reye-face[m].leye,
                                            face[m].cmouth-face[m].teye,
      140+face[m].leye,sy+256+face[m].teye,face[m].reye-face[m].leye,
                                           face[m].cmouth-face[m].teye);
/* left internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].center-face[m].leye,
                                            face[m].cmouth-face[m].teye,
      270+face[m].leye,sy+256+face[m].teye,face[m].center-face[m].leye,
                                           face[m].cmouth-face[m].teye);
/* bottom half: tnose to chin */
carea(sx,sy+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose,
      400,sy+256+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose);
line(sx,sy,sx+face[m].dx,sy,0);                                   /*top*/
line(sx+face[m].dx,sy,sx+face[m].dx,sy+face[m].dy,0);           /*right*/
line(sx+face[m].dx,sy+face[m].dy,sx,sy+face[m].dy,0);          /*bottom*/
line(sx,sy+face[m].dy,sx,sy,0);                                  /*left*/
line(sx,sy+face[m].teye,sx+face[m].dx,sy+face[m].teye,0);        /*teye*/
line(sx,sy+face[m].cmouth,sx+face[m].dx,sy+face[m].cmouth,0); /*cmouth*/
line(sx,sy+face[m].tnose,sx+face[m].dx,sy+face[m].tnose,0);     /*tnose*/
line(sx+face[m].leye,sy,sx+face[m].leye,sy+face[m].dy,0);        /*leye*/
line(sx+face[m].center,sy,sx+face[m].center,sy+face[m].dy,0); /*center*/
line(sx+face[m].reye,sy,sx+face[m].reye,sy+face[m].dy,0);        /*reye*/
ix = face[m].dx/2;
iy = face[m].dy/2;
printf("\n calculating gestalt for window 1.");
clear_cray();                    /* left half: whole head */
for (y=sy; y<sy+face[m].dy; y+=2)
 for (x=270; x<270+face[m].dx/2; x+=2)
  cray[1+(x-269)/2][1+(y-29)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win1x = (int) jr3d3;
ilist[0].win1y = (int) ir3d3;
printf(" win1x=%d win1y=%d",ilist[0].win1x,ilist[0].win1y);
```

```
printf("\n calculating gestalt for window 2.");
clear_cray();                    /* right half: whole head */
for (y=sy; y<sy+face[m].dy; y+=2)
 for (x=400; x<400+face[m].dx/2; x+=2)
  cray[1+(x-399)/2+face[m].dx/4][1+(y-29)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win2x = (int) jr3d3;
ilist[0].win2y = (int) ir3d3;
printf(" win2x=%d win2y=%d",ilist[0].win2x,ilist[0].win2y);
printf("\n calculating gestalt for window 3.");
clear_cray();                    /* top half: top to tnose */
for (y=sy+256; y<sy+256+face[m].tnose; y+=2)
 for (x=15; x<15+face[m].dx; x+=2)
  cray[1+(x-14)/2][1+(y-285)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win3x = (int) jr3d3;
ilist[0].win3y = (int) ir3d3;
printf(" win3x=%d win3y=%d",ilist[0].win3x,ilist[0].win3y);
printf("\n calculating gestalt for window 4.");
clear_cray();                           /* internal features */
for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
 for (x=140; x<140+face[m].reye; x+=2)
  cray[1+(x-139)/2][1+(y-285)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win4x = (int) jr3d3;
ilist[0].win4y = (int) ir3d3;
printf(" win4x=%d win4y=%d",ilist[0].win4x,ilist[0].win4y);
printf("\n calculating gestalt for window 5.");
clear_cray();                      /* left internal features */
for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
 for (x=270; x<270+face[m].dx/2; x+=2)
  cray[1+(x-269)/2][1+(y-285)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win5x = (int) jr3d3;
ilist[0].win5y = (int) ir3d3;
printf(" win5x=%d win5y=%d",ilist[0].win5x,ilist[0].win5y);
printf("\n calculating gestalt for window 6.");
clear_cray();                     /* bottom half: tnose to chin */
for (y=sy+256; y<sy+256+face[m].dy; y+=2)
 for (x=400; x<400+face[m].dx; x+=2)
  cray[1+(x-399)/2][1+(y-285)/2] = (double) 255 - brpixel(x,y);
cortran16();
ilist[0].win6x = (int) jr3d3;
ilist[0].win6y = (int) ir3d3;
printf(" win6x=%d win6y=%d",ilist[0].win6x,ilist[0].win6y);
return;
}
```

```
/*******************************************************************/
cont_enhance(m)
     int m; /* face number */
{
     int x,y,z;
  static_luts();
  setlut(RED,5);
  histeq(RED,5,face[m].leye+sx+1,face[m].beye+sy,
         face[m].dx/2 - 2,face[m].cmouth-face[m].beye);
  maplut(RED,5,0,0,256,256);
  linlut(RED,5);
  linlut(GREEN,5);
  linlut(BLUE,5);
  for (y=sy; y<sy+face[m].dy; y++) {       /* threshold result */
   for (x=sx; x<sx+face[m].dx; x++) {
    z = brpixel(x,y);                      /* leave dark areas but */
    if (z < 50) bwpixel(x,y,z);            /* make skin pure white */
    else bwpixel(x,y,255);
  }}
return;
}



/**********************************************************************/
scale(m)
     int m;
{
     int fact;
  if ((fx-sx)/150 > (fy-sy)/150) fact = 150/(fx-sx);
  else fact = 150/(fy-sy);
  if (fact > 1) {
   repzoom(sx,sy,fx-sx,fy-sy,sx,sy,200,200,fact,fact);
   face[m].dx = face[m].dx * fact; /* update face[m].lines by 'fact' */
   face[m].dy = face[m].dy * fact;
   face[m].leye = face[m].leye * fact;
   face[m].reye = face[m].reye * fact;
   face[m].teye = face[m].teye * fact;
   face[m].beye = face[m].beye * fact;
   face[m].tnose = face[m].tnose * fact;
   face[m].cmouth = face[m].cmouth * fact;
   face[m].center = face[m].center * fact;
   fx = (fx-sx)*fact + sx;   /* update fx,fy by 'fact' */
   fy = (fy-sy)*fact + sy;
  }
```

```
  aclear(0,0,512,sy,255);
  aclear(0,fy,512,480-fy,255);
  aclear(0,sy-1,sx,fy-sy+1,255);
  aclear(fx,sy-1,512-fx,fy-sy+1,255);
return;
}



/*************************************************************************/
int isolate(thresh,mode,size)   /* works on top half of screen only! */
    int thresh; /* threshold for detection of target */
    int mode;   /* 6 bit or 8 bit image */
    int size;   /* determines minimum size of target and affects speed. */
{            /* size is either 16 or 32 pixels. */
    int x,y,z;
sx = sy = fx = fy = -1;  /* Find top.    **********************************/
for (y=size-1; y <= 255; y=y+size){     /* This subroutine finds location */
 for (x = 0; x < 511; x=x+size){        /* of a moving object. If there is*/
  z = brpixel(x,y);                     /* no moving object, or it is too */
  if (mode == 6) z = z & 63;            /* small then (0) is returned. If */
  if (z >= thresh) {                    /* an object is found then sx,sy, */
   sy = y-(size-1);                     /* fx,fy are set and (1) is re-   */
   x = 512;                             /* turned. This is done so that   */
   y = 512;                             /* all future work done on a scene*/
}}}                                     /* is done on a greatly reduced   */
if (sy == -1) return(0);                /* area of the scene and hence is */
for (y=256-size; y>(sy+size-1); y=y-size){/* done faster. Thresh is set to*/
 for (x = 0; x <= 511; x=x+size){       /* high enough value to eliminate */
  z = brpixel(x,y);                     /* video noise but low enough to  */
  if (mode == 6) z = z & 63;            /* find small brightness differen-*/
  if (z >= thresh){                     /* ces that may occur between a   */
   fy = y + size-1; /* Find bottom.      * moving object and its bkgnd.   */
   x = 512;                             /**********************************/
   y = -1;
}}}
if (fy < (sy + size)) return(0);
for (x=size-1; x <= 511; x=x+size){  /* find left side */
 for (y = 0; y < 255; y=y+size){
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   sx = x - (size-1);
   x = y = 512;
}}}
```

```
if (sx == -1) return(0);
for (x = 512-size; x > (sx + size-1); x = x - size){
 for (y = 0; y < 255; y = y + size){  /* find right side */
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   fx = x + size-1;
   x = -1;
   y = 512;
}}}
if (fx < (sx + size)) return(0);
return(1);
}
```

```
/***************************************************************************/
#define A0 (short int)a0(i)    /* These are the transformations used in the  */
#define a0(i) (i & 0x003f)     /* feedback lut for the real time subtraction */
#define A1 (short int)a1(i)    /* demo. This software was created by using   */
#define a1(i) ((i & 0x0fc0) >> 6)  /* the toolbox program (see FG-100 user's */
#define D0(i) { data &= 0xffc0; data |= (i & 0x003f); }  /* manual chapt 7)  */
#define D1(i) { data &= 0xf03f; data |= ((i << 6) & 0x0fc0); } /**************/
#define INPUT 0x6000
#define abs(i) (((i) < 0) ? (-(i)) : (i))

xform1(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A1);
D0(abs(A1 - A0));
return((unsigned)data);
}

xform2(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A0);
D0(abs(A1 - A0));
return((unsigned)data);
}
```

```
/**************************************************************************/
afrm()        /* A completely Autonomous Face Recognition Machine (AFRM) */
{
 int cam;
 char t2[30],t3[30],stop,answer[1];
 register unsigned j;
stop = 'n';
printf("\n Select camera port (0,1 or 2) >");
scanf("%d",&cam);
while (stop == 'n') {
 cls();
 printf(" please wait...");
 rtsubtract(0);
 setcamera(cam);
 setlut(0,0);
 setinmux(6);
 for (j=0; j<0x1000; j++) write_lut(INPUT,j,xform2(j,read_lut(INPUT,j)));
 cls();
 printf(" looking for target.");
 snap(1);
 snap(1);
 while((isolate(8,6,32)) != 1) snap(1);
 printf("\n found target, acquiring 8 bit image.");
 initialize();
 setcamera(cam);
 waitvb();
 snap(1);
 nf = sx = sy = 0;
 fx = 511;
 fy = 255;   /* presently isolate() only looks for target in top */
 if (facemap() == 1) {     /* half so look for faces in top half */
  printf("\n found ");
  printf("%d",nf);
  if (nf == 1) printf(" face.");
  else printf(" faces.");
  facerec(2);
 }
 printf("\n Do you wish to stop? (Y/N) >");
 scanf("%s",answer);
 if (answer[0] == 'Y' || answer[0] == 'y') stop = 'y';
}
return;
}
```

```
/*************************************************************************/
rtransa()     /* modified from RTRANSA.FR written 06/27/85 by R. RUSSEL */
{
     int    s,j;
     double arg;
for (s=1; s<256; s++) {
 j = s - 128;
 arg = ((double)j*j)/(-6000.0);
 gauss[j+128] = exp(arg);
}
return;
}
```

```
/*************************************************************************/
rtransb()     /* modified from RTRANSB.FR written 06/28/85 by R. RUSSEL */
{             /* multiplys rinp by gaussian and puts in output */
     int i,j;
     double output[129];
for (i=1; i<129; i++) {
 output[i] = 0;
 for (j=1; j<129; j++) output[i] = output[i] + rinp[j]*gauss[j-i+128];
}
for (i=1; i<129; i++) rinp[i] = output[i];
return;
}
```

```
/******************************************************************************/
clear_cray()
{
     int x,y;
for (y=1; y<129; y++) for (x=1; x<129; x++) cray[x][y] = 0.0;
return;
}
```

```
/*************************************************************************/
cortran16()       /* Modified from CORTRAN16.FR 11/23/85 by R. RUSSEL */
{
     int j,i,iwinmax;
     double c,bmax,ir3d,jr3d;
for (i=1; i<iy+2; i++) {          /* see note.  Do 2D gestalt, rows first */
 for (j=1; j<129; j++) rinp[j] = cray[j][i];
 rtransb();
 for (j=1; j<129; j++) cray[j][i] = rinp[j];       /* put result into array */
}
for (j=1; j<ix+2; j++) {           /* see note.           Now do columns */
 for (i=1; i<129; i++) rinp[i] = cray[j][i];
 rtransb();
 for (i=1; i<129; i++) cray[j][i] = rinp[i];
}
bmax = 0.0;                  /* Columns completed. Find location of max value */
ir3d = 64.0;                         /* define preset values for a zero array */
jr3d = 64.0;
for (i=1; i<iy+2; i++) {
 for (j=1; j<ix+2; j++) {
  c = cray[j][i];
  if ( c > bmax) {                         /* note: only go to ix,iy in array */
   bmax = c;                               /* because beyond that is all zero */
   ir3d = (double)i;
   jr3d = (double)j;
}}}
iwinmax = iy;                                                     /* scale */
if (ix > iwinmax) iwinmax = ix;
ir3d3 = ir3d*(128.0/(double)iwinmax) + 0.5;
jr3d3 = jr3d*(128.0/(double)iwinmax) + 0.5;
return;
}


/*************************************************************************/
del()
{
printf("\n\n           Deleting files with reserved names.");
system("delete bnorm.img;*");  /* these names are reserved for facefinder */
system("delete orig.img;*");
return;
}
/*************************************************************************/
```

```
/***************************************************************************/
facerec(version)
int version;
{
 char ch[2],t2[30],t3[30],t4[30],t5[30];
 int l,m,n,p,dx,dy;
if (nf != 0) {
 cls();
 printf(" trying to recognize faces found...");
 for (m=1; m<nf+1; m++) {
  t2[0] = '\0';                    /* create file names for face # m */
  t4[0] = '\0';
  strcat(t2,"bnorm.img;\0");
  strcat(t4,"orig.img;\0");
  t3[0] = m + '0';
  t3[1] = '\0';
  strcat(t2,t3);
  strcat(t4,t3);
  sx = 60;
  sy = 30;
  printf("\n %s",t2);
  sclear(0,1);
  readim(sx,sy,200,200,t2,"nocomm");    /* display bright_norm face */
  l = sy;
  while(brpixel(sx,l) != 0) l++;              /* get fx,fy values */
  fy = l - 1;
  l = sx;
  while(brpixel(l,sy) != 0) l++;
  fx = l - 1;
  dx = fx - sx;
  dy = fy - sy;
  cont_enhance(m);
  scale(m);
  text(70,10,0,1,0,t2);
  gestalt(m);                      /* gestalt values put in ilist[0] */
  initialize();
  sclear(0,1);
  readim(200,sy,200,200,t4,"nocomm");        /* display original face */
  text(200,10,0,1,200,t4);
  if (version == 1) {
   printf("\n Save in dbase? (Y/N) >");
   scanf("%s",ch);
   if (ch[0] == 'y' || ch[0] == 'Y') {
     printf("\n enter name of subject (up to 10 letters) \n>");
     scanf("%s",t3);
     p = 0; /* highest existing version # for this subject */
     for (n=1; n<k+1; n++) {
       if (strcmp(ilist[n].name,t3) == 0) {
         p = ilist[n].num;
     } }
     k = k + 1;
     p = p + 1;
     ilist[k].name[0] = '\0';
     strcat(ilist[k].name,t3);
     ilist[k].num = p;
```

```
        ilist[k].win1x = ilist[0].win1x;
        ilist[k].win1y = ilist[0].win1y;
        ilist[k].win2x = ilist[0].win2x;
        ilist[k].win2y = ilist[0].win2y;
        ilist[k].win3x = ilist[0].win3x;
        ilist[k].win3y = ilist[0].win3y;
        ilist[k].win4x = ilist[0].win4x;
        ilist[k].win4y = ilist[0].win4y;
        ilist[k].win5x = ilist[0].win5x;
        ilist[k].win5y = ilist[0].win5y;
        ilist[k].win6x = ilist[0].win6x;
        ilist[k].win6y = ilist[0].win6y;
        t5[0] = '\0';
        strcat(t5,"[face.dbase]\0");
        strcat(t5,ilist[k].name);
        strcat(t5,".img;\0");
        saveim(200,sy,dx,dy,0,t5,"nocomm");
        writefile("[face.dbase]others.dat;1",ilist,k);
      }
      else {
       if (m < nf) {
         printf("\n Forget about rest of faces and return to main menu? (Y/N) >");
         scanf("%s",ch);
         if (ch[0] == 'y' || ch[0] == 'Y') {
           return;
         }
       }
      }
    }
    recognize(0);                  /* pass in gestalt values of ilist[0] */
    delete(t2);
    delete(t4);
    cls();
  }}
  else {
     printf("\n face not found.");
     prtc();
  }
  nf = sx = sy = 0;
  fx = fy = 511;
  return;
  }
```

```
/***************************************************************************/
static int results1[257][5];
static int list[101]; /* list of id#s ordered by distances in list2 */
static double t[101],list2[101]; /* total distances (for all windows) */
static double v[101][7]; /* v[id][w] = distance from person #id to
                 unknown person for window #w  (Russel, 1985:4-40a) */



/***************************************************************************/
recognize(num)    /*from REMID.FR 06/03/86 by R. Russel */
 int num; /* the position in ilist[] of gestalt values to use. */
{
char t8[30];
double gix,giy,gux,guy,sigix,sigiy,a,b,c,most;
int id,w,m,n,j,confid,test;
double p[7] = {10.0,1.0,1.5,2.0,3.0,1.5,1.0};  /* window performance factors
                (update after training and testing with sufficient samples */
                                  /* note: p[0] is used for total of factors */
printf("\n\n\n Now trying to recognize subject in top half of screen.\n");
printf("\n Presently trained with %d subjects.",(i/4));

w = 1;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win1x + tlist[m+1].win1x + tlist[m+2].win1x
                 + tlist[m+3].win1x) )/4.0;
   giy = ( (double) (tlist[m].win1y + tlist[m+1].win1y + tlist[m+2].win1y
                 + tlist[m+3].win1y) )/4.0;
   gux = (double) ilist[num].win1x;
   guy = (double) ilist[num].win1y;
   sigix = ( (double) (abs(gix-tlist[m].win1x)*abs(gix-tlist[m].win1x)+
              abs(gix-tlist[m+1].win1x)*abs(gix-tlist[m+1].win1x)+
              abs(gix-tlist[m+2].win1x)*abs(gix-tlist[m+2].win1x)+
              abs(gix-tlist[m+3].win1x)*abs(gix-tlist[m+3].win1x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win1y)*abs(giy-tlist[m].win1y)+
              abs(giy-tlist[m+1].win1y)*abs(giy-tlist[m+1].win1y)+
              abs(giy-tlist[m+2].win1y)*abs(giy-tlist[m+2].win1y)+
              abs(giy-tlist[m+3].win1y)*abs(giy-tlist[m+3].win1y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}
```

```
v = 2;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win2x + tlist[m+1].win2x + tlist[m+2].win2x
                     + tlist[m+3].win2x) )/4.0;
   giy = ( (double) (tlist[m].win2y + tlist[m+1].win2y + tlist[m+2].win2y
                     + tlist[m+3].win2y) )/4.0;
   gux = (double) ilist[num].win2x;
   guy = (double) ilist[num].win2y;
   sigix = ( (double) (abs(gix-tlist[m].win2x)*abs(gix-tlist[m].win2x)+
                 abs(gix-tlist[m+1].win2x)*abs(gix-tlist[m+1].win2x)+
                 abs(gix-tlist[m+2].win2x)*abs(gix-tlist[m+2].win2x)+
                 abs(gix-tlist[m+3].win2x)*abs(gix-tlist[m+3].win2x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win2y)*abs(giy-tlist[m].win2y)+
                 abs(giy-tlist[m+1].win2y)*abs(giy-tlist[m+1].win2y)+
                 abs(giy-tlist[m+2].win2y)*abs(giy-tlist[m+2].win2y)+
                 abs(giy-tlist[m+3].win2y)*abs(giy-tlist[m+3].win2y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}

w = 3;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win3x + tlist[m+1].win3x + tlist[m+2].win3x
                     + tlist[m+3].win3x) )/4.0;
   giy = ( (double) (tlist[m].win3y + tlist[m+1].win3y + tlist[m+2].win3y
                     + tlist[m+3].win3y) )/4.0;
   gux = (double) ilist[num].win3x;
   guy = (double) ilist[num].win3y;
   sigix = ( (double) (abs(gix-tlist[m].win3x)*abs(gix-tlist[m].win3x)+
                 abs(gix-tlist[m+1].win3x)*abs(gix-tlist[m+1].win3x)+
                 abs(gix-tlist[m+2].win3x)*abs(gix-tlist[m+2].win3x)+
                 abs(gix-tlist[m+3].win3x)*abs(gix-tlist[m+3].win3x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win3y)*abs(giy-tlist[m].win3y)+
                 abs(giy-tlist[m+1].win3y)*abs(giy-tlist[m+1].win3y)+
                 abs(giy-tlist[m+2].win3y)*abs(giy-tlist[m+2].win3y)+
                 abs(giy-tlist[m+3].win3y)*abs(giy-tlist[m+3].win3y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}
```

```
w = 4;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win4x + tlist[m+1].win4x + tlist[m+2].win4x
                     + tlist[m+3].win4x) )/4.0;
   giy = ( (double) (tlist[m].win4y + tlist[m+1].win4y + tlist[m+2].win4y
                     + tlist[m+3].win4y) )/4.0;
   gux = (double) ilist[num].win4x;
   guy = (double) ilist[num].win4y;
   sigix = ( (double) (abs(gix-tlist[m].win4x)*abs(gix-tlist[m].win4x)+
                 abs(gix-tlist[m+1].win4x)*abs(gix-tlist[m+1].win4x)+
                 abs(gix-tlist[m+2].win4x)*abs(gix-tlist[m+2].win4x)+
                 abs(gix-tlist[m+3].win4x)*abs(gix-tlist[m+3].win4x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win4y)*abs(giy-tlist[m].win4y)+
                 abs(giy-tlist[m+1].win4y)*abs(giy-tlist[m+1].win4y)+
                 abs(giy-tlist[m+2].win4y)*abs(giy-tlist[m+2].win4y)+
                 abs(giy-tlist[m+3].win4y)*abs(giy-tlist[m+3].win4y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}

w = 5;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win5x + tlist[m+1].win5x + tlist[m+2].win5x
                     + tlist[m+3].win5x) )/4.0;
   giy = ( (double) (tlist[m].win5y + tlist[m+1].win5y + tlist[m+2].win5y
                     + tlist[m+3].win5y) )/4.0;
   gux = (double) ilist[num].win5x;
   guy = (double) ilist[num].win5y;
   sigix = ( (double) (abs(gix-tlist[m].win5x)*abs(gix-tlist[m].win5x)+
                 abs(gix-tlist[m+1].win5x)*abs(gix-tlist[m+1].win5x)+
                 abs(gix-tlist[m+2].win5x)*abs(gix-tlist[m+2].win5x)+
                 abs(gix-tlist[m+3].win5x)*abs(gix-tlist[m+3].win5x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win5y)*abs(giy-tlist[m].win5y)+
                 abs(giy-tlist[m+1].win5y)*abs(giy-tlist[m+1].win5y)+
                 abs(giy-tlist[m+2].win5y)*abs(giy-tlist[m+2].win5y)+
                 abs(giy-tlist[m+3].win5y)*abs(giy-tlist[m+3].win5y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}
```

```
w = 6;
for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   gix = ( (double) (tlist[m].win6x + tlist[m+1].win6x + tlist[m+2].win6x
                     + tlist[m+3].win6x) )/4.0;
   giy = ( (double) (tlist[m].win6y + tlist[m+1].win6y + tlist[m+2].win6y
                     + tlist[m+3].win6y) )/4.0;
   gux = (double) ilist[num].win6x;
   guy = (double) ilist[num].win6y;
   sigix = ( (double) (abs(gix-tlist[m].win6x)*abs(gix-tlist[m].win6x)+
                  abs(gix-tlist[m+1].win6x)*abs(gix-tlist[m+1].win6x)+
                  abs(gix-tlist[m+2].win6x)*abs(gix-tlist[m+2].win6x)+
                  abs(gix-tlist[m+3].win6x)*abs(gix-tlist[m+3].win6x)) )/4.0;
   sigix = sqrt(sigix);
   if (sigix < .5) sigix = .5;
   sigiy = ( (double) (abs(giy-tlist[m].win6y)*abs(giy-tlist[m].win6y)+
                  abs(giy-tlist[m+1].win6y)*abs(giy-tlist[m+1].win6y)+
                  abs(giy-tlist[m+2].win6y)*abs(giy-tlist[m+2].win6y)+
                  abs(giy-tlist[m+3].win6y)*abs(giy-tlist[m+3].win6y)) )/4.0;
   sigiy = sqrt(sigiy);
   if (sigiy < .5) sigiy = .5;
   a = (gix-gux)*(gix-gux)/(4*sigix*sigix);
   b = (giy-guy)*(giy-guy)/(4*sigiy*sigiy);
   c = a + b;
   v[id][w] = exp(-1.0*c/1.4) * p[w];
}

for (id=1; id<(i/4)+1; id++) {
   t[id] = 0.000000001;
   for (w=1; w<7; w++) {
     t[id] += v[id][w];
   }
   t[id] = t[id]/p[0]; /* max t[id] = 1.0 when distance from id to unknown */
}                                                 /* individual = 0.0 */

/* now have all distances ordered by id#, need to order id#s by distance */
for (m=1; m<101; m++) {
   list[m] = 0;
   list2[m] = 0.000000001;
}
for (m=1; m<(i/4)+1; m++) {
   most = 0.000000001;
   for (j=1; j<(i/4)+1; j++) {
      if (t[j] > most) {
         most = t[j];
         n = j;
      }
   }
   list[m] = n;           /* id # */
   list2[m] = t[n];       /* distance */
   t[n] = 0.000000001;
}
```

```
/* now have ordered list of candidates, need to display them */
test = 0;
if (list2[1] > 0.001) {
   printf("\n\n                Candidate      Distance");
/*   printf("   Confidence");   */
   for (m=1; m<(i/4)+1; m++) {
      if (list2[m] > 0.001) {
         if (m == 1) {
            printf("\n 1st Choice: ");
            t8[0] = '\0';
            strcat(t8,"[face.dbase]\0");
            strcat(t8,tlist[list[1]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(50,286,200,200,t8,"nocomm");
            text(50,266,0,1,200,tlist[list[1]*4 - 3].name);
            test = 1;
         }
         if (m == 2) {
            printf("\n 2nd Choice: ");
            t8[0] = '\0';
            strcat(t8,"[face.dbase]\0");
            strcat(t8,tlist[list[2]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(200,286,200,200,t8,"nocomm");
            text(200,266,0,1,200,tlist[list[2]*4 - 3].name);
            test = 2;
         }
         if (m == 3) {
            printf("\n 3rd Choice: ");
            t8[0] = '\0';
            strcat(t8,"[face.dbase]\0");
            strcat(t8,tlist[list[3]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(350,286,200,200,t8,"nocomm");
            text(350,266,0,1,200,tlist[list[3]*4 - 3].name);
            test = 3;
         }
         if (m == 4) printf("\n     Others: ");
         if (m > 4) printf("\n              ");
/*         confid = based on distance of this candidate and
                        distances to next candidates */
         printf("%11s     %f",tlist[list[m]*4 - 3].name,list2[m]);
 /*        printf("      %d",confid);*/
      }
      else m=200;
   }
}
```

```
if (test == 0) {
   printf("\n\n Could not find any close enough candidates.");
   printf("\n The computer has never seen this person before.");
}
if ( test < 3 && test != 0) {
  printf("\n\n Could not find any more close enough candidates so");
  if (test == 1) printf("\n only displayed 1 picture.");
  else printf("\n only displayed 2 pictures.");
}
prtc();
return;
}
/******************************************************************************/
/* End of program */
```

ITEX-100 Subroutines used by FACE.C

ACLEAR
BLUR
BRPIXEL
BWPIXEL
CAREA
CIRCLE
FILL
GRAB
HISTEQ
INITIALIZE
INITREGS
LINE
LINLUT
LOPASS
MAPLUT
MAREA
OPAREA
READIM
RECTANGLE
REPZOOM
RHLINE
RPIXEL
RTSUBTRACT
SAVEIM
SCLEAR
SETCAMERA
SETHDW
SETINMUX
SETLUT
SETREG
SHARPEN
SNAP
STATIC_LUTS
STOPGRAB
SWAP6
TEXT
WAITVB
WHLINE
WPIXEL

Appendix C

Autonomous Face Recognition Machine

User's Manual

Table of Contents

Introduction

The information presented in this manual is divided into two parts: Chapter 1 gives enough information for a casual user to operate the AFRM, and Chapter 2 gives information needed to modify, initialize and re-host the AFRM.

User friendliness was a primary concern when writing the code for the AFRM. The AFRM contains self-explanatory menu options, it tells the user exactly what is required from each keyboard (user) entry and it is fault tolerant. The AFRM code has been commented on in detail and uses only simple C programming techniques (no nonsense like pointers to arrays of pointers). The goal was to write the code as efficiently as necessary, and then as readable as possible.

It is hoped that future modifications to the AFRM will strive to maintain an easy user interface and minimum additions to this manual.

I. Operation

Logging On and Off

The easiest way to get to know the AFRM is to sit down and use it. It is located on the Micro-VAX II designated SMV2A in the AFIT Signal Processing Lab. To run the AFRM, log onto SMV2A using the username FACE, no password is required. The AFRM will run automatically and will perform several seconds of hardware and software initialization and will then present the main menu. When you are done using the AFRM, return to this menu and select the QUIT option. This will get you out of the program and automatically logout.

Two Things You Should Not Do

1. The AFRM needs to create temporary files now and then as a normal part of its operation. It will delete these files as soon as they are no longer needed. Since these files are created and deleted without informing the user, the user should avoid saving files with these temporary-file names. Never save faces in files named:

BNORM.IMG
ORIG.IMG

At some un-announced point in time YOU WILL LOSE THEM.

2. The AFRM has been designed to be fault tolerant. You can enter anything you want, at any prompt you want, and the AFRM should handle it. The AFRM will inform you if your input is invalid. The only entries not allowed are CTRL-C

and CTRL-Y which terminate the program without going to the main menu option QUIT. These are not allowed because the AFRM will not be able to save updated database files and because the user will be allowed into the FACE account where he shouldn't be (the AFRM won't cause an automatic logout).

During normal AFRM operation this is not a concern for the user because the CTRL-C and CTRL-Y entries are disabled by a protection scheme described in Chapter 2. It is only mentioned here to remind Special Users (those who modify, install or initialize the system) to re-install the protection scheme when they are done and to make them aware of the consequences of CTRL keys when protection is not in place.

Image Acquisition

There are several ways to input images into the AFRM and there is a sub-menu for all the options. This sub-menu is obtained by selecting main menu option #1. Most of the menu options are self-explainatory and so minimum detail is given here.

0: Return to Main Menu

1: Stationary Target - Allows acquisition of a 512 X 480 image from the camera.

2: Moving Target - Acquires a background scene from the camera (nobody in it), then acquires a second scene (with subject). The AFRM will provide the rectangular area that is different in the two scenes. This target area is all that is processed by the face locator (if locator is selected) and so the face locator will be faster than it would be for a full size scene.

3: Load From Memory - Allows user to load a previously stored image (for example, an image stored in the user's personal account on SMV2A).

4: Save in [FACE] - You can save images in this account if desired but please reserve the space in this account for images that are useful to everybody. If you only want the image for yourself then login to your account and save the image using TEST100.

5: Set Camera Port - The default port is (0) and this allows use of the Dage camera. The two General Electric cameras are connected to ports (1) and (2).

6: Camera Check - Allows continuous acquisition of images so you can position and focus the camera.

7: Re-initialize Hardware - Go back to default camera port, clear the screen, etc.

Face Location

Main menu option #2 runs the face location algorithm. This algorithm will look for faces in the image on the screen and save all it finds to temporary files. There is an option to sharpen the scene that is normally not needed but sometimes helps the face finding process. This sharpening option may be removed in the future.

Gestalt and Identification

Main menu option #3 only works after a face(s) was found by option #2. If no face(s) was found then this option will return to the main menu. This option runs the gestalt algorithm on the first face found by option #2. Then it runs the recognition algorithm on that face. During recognition the user is allowed to save the face and its gestalt data in the database. There is no other time when a new face and its gestalt data are available for saving in the database so save it NOW if you want it, otherwise you will have to Gestalt it again later (faces are easily deleted from the database if you change your mind later). If more than one face was found in option #2 than all faces will be gestalted and identified in the order found.

Care and Feeding of the Database

Several main menu options are discussed in this section because they all have something to do with using and changing the database.

4: Display Contents of Database

5: Delete a Subject - To "delete a subject", means to delete the training file for this subject. The actual images and gestalt values can still be saved in the .IMG section (files it is not trained on) and the AFRM can be re-trained with this subject later. You may also delete this subject from the database altogether if desired.

6: Delete an Image - This option allows the deletion of single images (files that the AFRM is not trained on) from the database.

7: Train - This allows the user to train the database with 4 files from the .IMG section of the database. The files must all have the same name and must have different version numbers. To exit this option at any time, enter a negative version number.

Fault tolerance is really evident in this section of the AFRM because it is so important to maintain a correct database. The AFRM constantly checks user inputs for validity and gives out pertinent information when it finds a mistake. For example, suppose it is decided to train the AFRM with the name Smith, version numbers 1, 2, 3, and 4. The AFRM will verify that the name you enter exists in the .IMG section and that it does not exist in the trained section. It will verify that files exist for all the versions you type in and that you have not typed the same version number more than once. If you make a mistake and wish to exit to the main menu, you are allowed to do so at any time.

Demonstration

This section is for users who are already familiar with the operation of the other main menu items. The demonstration option (#8) provides a menu with the following options:

Identifying an Individual

This option allows the user to demonstrate the recogni-

tion capabilities of the AFRM by selecting an un-trained image from the database and asking the AFRM who it is. This option is also used to obtain recognition scores so that the AFRM can be evaluated.

The Total System

This option allows the user to run all AFRM algorithms together starting at image acquisition and ending with recognition. This option is run as follows:

1. Run the "Camera Ckeck" option and set the camera up to take a full body picture of a standing person. Then ensure nobody is standing in the field of view of the camera.

2. Select the total system option.

3. Select the camera port desired. After a couple seconds, the screen will go black as the camera continually acquires images and the itex board applies real-time subtraction.

4. When the screen is black, have a subject walk into the field of view of the camera, turn and stare at the camera, and stand still for a few seconds. As soon as the AFRM "sees" the subject it will snap a picture and begin to look for a face. (There is no sharpening option to worry about here.) If a face is found, then the AFRM will gestalt it and try to recognize the individual. (There is no save option here.)

5. After recognition, the user will be asked if the whole process should repeat.

Other Programs

There are other programs associated with the development of the AFRM that may be useful to some users. These programs are found in the following directory:

dua2:[llambert.cdir]

The programs can be run by typing the following:

run [llambert.cdir]program_name

If you don't have an account on SMV2A where you can login and run these programs, then login as USER, no password required.

The following programs are available and are described in Lambert's 1987 masters thesis, "Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine".

Sub_Demo.exe
MTI.exe
Bright.exe
Graph.exe
Face_Sig.exe

II. Technical Details

Where is Everything?

The executable AFRM program is located in the directory dua2:[face] and is called FACE.EXE. The database files are located in a sub-directory called dua2:[face.dbase]. There are two database files, called TRAIN.DAT;1 (for the trained gestalt files), and OTHERS.DAT;1 (for the un-trained files). The actual images of people stored in the database are also located in this sub-directory.

The source code for the AFRM, called FACE.C, is located in dua2:[llambert.cdir]. A good example of code for running itex routines is TEST100.C, located in dua0:[iti100.itex].

Startup

If the AFRM is not running properly, has been changed, is being hosted on another computer or for some other reason needs to be started up from scratch, the following steps must be done:

1. Ensure that FACE.EXE is located in a directory called [FACE] and that the database files are put in [FACE.DBASE]. If you use any other directory names, then modify the source code that specifically calls out these names and re-compile.

2. Ensure that the database files TRAIN.DAT;1 and OTHERS.DAT;1 exist. They can be created by putting an asterisk (*) into each file. The asterisk is the EOF indicator looked for by the AFRM when it reads these files.

There does not have to be any gestalt data in the files to start with.

3. Ensure that the protection scheme described in the next section is in place.

Protection

In order to protect the AFRM and its database from accidental changes/erasures, a login command file has been setup that automatically runs the AFRM upon login and automatically logs out when the AFRM stops. It also protects the database files by turning off the CTRL-Y and CTRL-C functions before running the AFRM.

In order to get around the protection, login to SMV2A as FACE and immediately start hitting CTRL-C. This will terminate the LOGIN.COM file as soon as it begins. Now you will be logged on and can do anything you want to the [FACE] and [FACE.DBASE] directory contents.

LOGIN.COM contains the following commands:

```
$show quota
$set nocontrol=y
$define/user_mode sys$input sys$command:
$run face
$lo*gout :== logout/full
$lo
```

Modification

If a change is needed in the AFRM then the C source code (FACE.C) has to be edited, re-compiled and linked to the appropriate libraries. The following commands are needed to accomplish this:

1. EDIT FACE.C
2. CC FACE.C
3. @L FACE

The third command runs a command file called L.COM which identifies all the appropriate libraries for you (so you don't have to do all that typing). L.COM and an associated file called OPTIONS_FILE.OPT are located in:

dua2:[llambert.cdir]

and should be copied into your own directory for use. The contents of these files are as follows:

L.COM

```
link 'P1',dua0:[iti100.itex]itex100/library,
          dua0:[iti100.toolbox]toolbox/library,
          dua0:[iti100.vms]vms100/library,
          dua2:[llambert.cdir]options_file.opt
```

OPTIONS_FILE.OPT

```
SYS$SHARE:VAXCRTL.EXE/SHARE
```

A modification that may be necessary in the future is a change to the declared size of the arrays in the AFRM. The AFRM is presently set to handle up to 100 subjects in the training file (400 gestalt sets, tlist[400]) and 100 images in the non-trained file (100 gestalt sets, ilist[100]).

Appendix D

Gestalt Files

The top line shown in these files (heading) is not present in the actual AFRM files. The asterisk (*) at the bottom is the EOF looked for by the AFRM.

NAME	VERSION	1X	1Y	2X	2Y	3X	3Y	4X	4Y	5X	5Y	6X	6Y
llambert	1	23	52	42	67	32	44	32	61	21	57	36	99
llambert	2	24	52	42	62	32	45	32	61	22	57	35	96
llambert	3	24	53	46	69	35	44	35	59	21	56	40	99
llambert	4	23	52	45	59	34	43	34	59	23	58	36	97
efretheim	1	21	62	43	64	32	51	32	60	21	58	34	94
efretheim	2	23	55	43	60	34	45	32	60	21	58	34	94
efretheim	3	19	58	39	66	31	48	31	62	19	58	33	95
efretheim	4	23	56	43	60	34	47	34	58	21	56	36	92
mkabrisky	1	20	66	42	69	31	51	31	60	20	60	31	97
mkabrisky	2	22	67	45	63	33	51	33	59	22	61	32	96
mkabrisky	3	24	63	50	67	37	50	37	59	22	59	37	98
mkabrisky	4	23	69	50	69	38	50	38	61	23	59	36	101
mdrylie	1	27	69	50	71	39	53	39	62	27	62	39	101
mdrylie	2	28	65	54	72	42	54	42	61	28	61	42	98
mdrylie	3	29	66	55	73	44	53	44	60	29	57	42	99
mdrylie	4	29	66	55	71	42	53	42	62	29	60	40	99
ecrawford	1	24	62	48	68	38	50	38	60	24	58	40	98
ecrawford	2	25	62	48	68	39	50	39	62	25	58	39	97
ecrawford	3	25	65	50	69	38	52	38	63	25	59	38	97
ecrawford	4	25	65	48	67	38	53	36	63	25	61	34	97
mlambert	1	24	60	43	66	38	47	34	72	23	70	34	100
mlambert	2	28	62	55	69	43	50	43	71	28	74	43	102
mlambert	3	25	79	54	79	45	58	41	78	25	81	38	106
mlambert	4	26	69	50	68	45	50	45	76	26	74	42	105
mmayo	1	24	49	49	59	39	39	39	59	24	57	41	98
mmayo	2	25	48	50	58	39	39	39	60	25	56	43	97
mmayo	3	25	48	48	56	37	41	37	60	23	58	39	97
mmayo	4	27	48	56	48	45	37	45	60	27	58	45	99
jsillart	1	27	59	59	62	44	54	44	59	27	59	44	94
jsillart	2	27	64	59	66	44	54	44	59	27	59	47	98
jsillart	3	27	64	59	64	44	54	44	62	27	62	44	96
jsillart	4	30	69	57	66	44	54	44	64	27	62	44	98

dlambert	1	29	52	56	54	47	38	47	63	31	58	47	97
dlambert	2	29	43	63	58	46	36	51	60	34	58	53	99
dlambert	3	28	43	59	52	45	36	47	59	31	57	50	100
dlambert	4	28	47	56	56	44	37	44	63	30	61	47	98
gdawson	1	25	54	50	56	39	41	39	62	25	60	37	97
gdawson	2	24	62	51	62	40	44	40	60	24	57	35	99
gdawson	3	24	65	52	63	41	48	41	59	26	59	33	98
gdawson	4	21	58	51	62	36	45	41	62	26	60	36	92
bgeorge	1	28	72	54	74	41	51	41	64	28	67	41	97
bgeorge	2	26	59	49	72	36	46	41	61	28	61	38	97
bgeorge	3	25	54	52	64	37	39	37	66	27	66	34	98
bgeorge	4	26	69	51	74	41	54	38	67	26	67	41	97
srogers	1	29	69	53	69	43	51	43	61	29	61	43	96
srogers	2	28	64	51	69	44	51	41	59	26	59	44	97
srogers	3	26	71	52	71	42	52	42	65	26	65	42	97
srogers	4	26	66	47	71	38	52	38	62	26	62	38	95
druck	1	28	63	51	67	42	54	42	61	28	58	42	95
druck	2	25	66	53	69	39	55	39	62	25	62	39	98
druck	3	27	62	53	66	41	53	41	59	27	59	41	94
druck	4	26	67	49	72	40	56	37	63	26	63	40	98

*

NAME	VERSION	1X	1Y	2X	2Y	3X	3Y	4X	4Y	5X	5Y	6X	6Y
llambert	1	22	50	43	67	32	43	34	61	22	56	36	96
efretheim	1	23	55	43	58	32	45	32	60	21	56	34	92
mkabrisky	1	25	67	47	67	35	49	35	60	25	60	35	96
ecrawford	1	24	65	49	67	39	51	39	61	24	57	39	98
mdrylie	1	29	68	55	73	42	53	42	62	29	60	42	99
mmayo	1	26	50	54	66	40	42	40	60	24	56	46	98
mlambert	1	26	67	52	72	45	46	41	74	26	74	41	106
jsillart	1	27	62	59	66	44	54	44	59	27	59	42	96
dlambert	1	27	45	58	56	45	36	47	65	29	58	52	99
gdawson	1	24	51	51	57	40	42	40	57	24	57	38	95
bgeorge	1	28	68	50	70	40	53	40	63	28	63	40	95
srogers	1	29	67	53	69	43	51	43	59	27	59	45	96
druck	1	25	62	53	69	41	55	41	64	27	62	41	96

*

Appendix E

Description of Brightness Normalization

The brightness normalization algorithm, shown on page E-4 is used to preprocess images fed to the AFRM. This appendix describes how the code implements the equation descri in Chapter 3, and shows the effects that the algorithm has on various scenes.

The algorithm reads the original image into the array "PIC" and puts the normalized image into the array "NORM". The sum of all pixels in a given neighborhood is called "NEIGH" and the average, "AVG" is this sum divided by the number of pixels in the neighborhood; in this case 81 (9X9). In order to speed up the program, it is noted that adjacent neighborhoods have common pixels. Each new neighborhood uses 72 of the previous neighborhood's pixels and 9 new pixels in its sum (NEIGH) so instead of adding up 81 pixels for each NEIGH value, 9 pixel values are subtracted from the previous sum and 9 pixel values are added to the previous sum. This reduces the total number of computations needed to perform the algorithm. Recognizing that each new set of 9 pixel values is a column of pixels, the computations can be further reduced by calculating the sums for all the columns first. In this way a new NEIGH value is the old value plus one column value minus another column value. This alone does not speed up the algorithm though because the column values still have to be computed. The speed up comes when the column computation is sped up. (just as we went horizontally across the screen adding 1 col and subtracting 1 col, we will update columns in the vertical direction by adding 1 pixel and sub-

tracting 1 pixel). The total number of computations has now been reduced from 512X512X81 (21 million per image) addition operations to approximately 512X512X4 (1 million) additions, allowing the algorithm to process an image in 8 seconds. The lines that perform the "speed processing" are lines 27 to 39 in the code.

The pixel that gets modified in each neighborhood is the center pixel for that neighborhood, as shown by lines 42 and 48 of the code. ([x,y] is the corner of a neighborhood so [x+4,y+4] is the center). This pixel is called "PIX". The equation discussed in Chapter 3 is implemented in the four lines numbered 41 through 44. Reading the image into PIC and writing NORM out to the screen (lines 23-25 and 50-52) adds about 6 seconds to the total process.

Figures E-1 through E-7 show the type of processing that can be accomplished by this alogirthm with various modifications to the equation in line 44. In all cases, the top half of the figure is a graph of the brightness variations along the black line indicated in the bottom half of the figure. Some figures show two images side by side. In these figures, the left half of the figure shows the original image and the right half shows the processed image.

```c
/**********************************************************************
*       BRIGHT.C : Brightness normalization algorithm              *
*                      will process whatever is on monitor.        *
*      Author : Laurence C. Lambert - 1987                        *
**********************************************************************/
#include "sys$library:stdio.h"
#include "dua0:[iti100.itex]stdtyp.h"
#include "dua0:[iti100.itex]itex100.h"
struct array{
       int data[512];
};
static struct array pic[512],norm[512];
static int col[512];
/***********************************************************************/
main()
{
    unsigned    base = 0x1600;
    long        mem = 0x200000L;
    int         flag = 1,block = 8;
    int         pix,avg,diff,neigh,x,y,i,j;
sethdw(base,mem,flag,block);
printf(" takes about 15 seconds to process. please wait...");
for (y=0; y<480; y++) {  /* read from video memory */
 rhline(0,y,512,pic[y].data);
}
y = 0;
for (i=0; i<512; i++) {
 col[i] = 0;              /*  setup all columns for first y value */
 for (j=y; j<y+9; j++) col[i] += pic[j].data[i];
}
for (y=1; y<471; y++) {
 for (i=0; i<512; i++) {  /* now all columns calculated faster  */
  col[i] += (pic[y+8].data[i] - pic[y-1].data[i]);
 }
 x = 0;
 neigh = 0;                          /* setup first neighborhood */
 for (i=x; i<x+9; i++) neigh += col[i];
 for (x=1; x<503; x++) { /* now all other neigh are calc faster */
  neigh += (col[x+8] - col[x-1]);

  avg = neigh/81;   /* these four lines are the heart of it all */
  pix = pic[y+4].data[x+4];     /* neighborhood size = 9x9      */
  diff = pix - avg;             /* center size = 1              */
  pix = 128 + diff;
                                    /* for awesome effects try: */
  if (pix < 0) pix = 0;             /*  other sizes,            */
  if (pix > 255) pix = 255;         /*  pix=128+multiplier*diff,*/
  norm[y+4].data[x+4] = pix         /*  thresholding result,    */
}}                                  /*  etc...                  */
for (y=0; y<480; y++) {
 whline(0,y,512,norm[y].data);
}
}
```

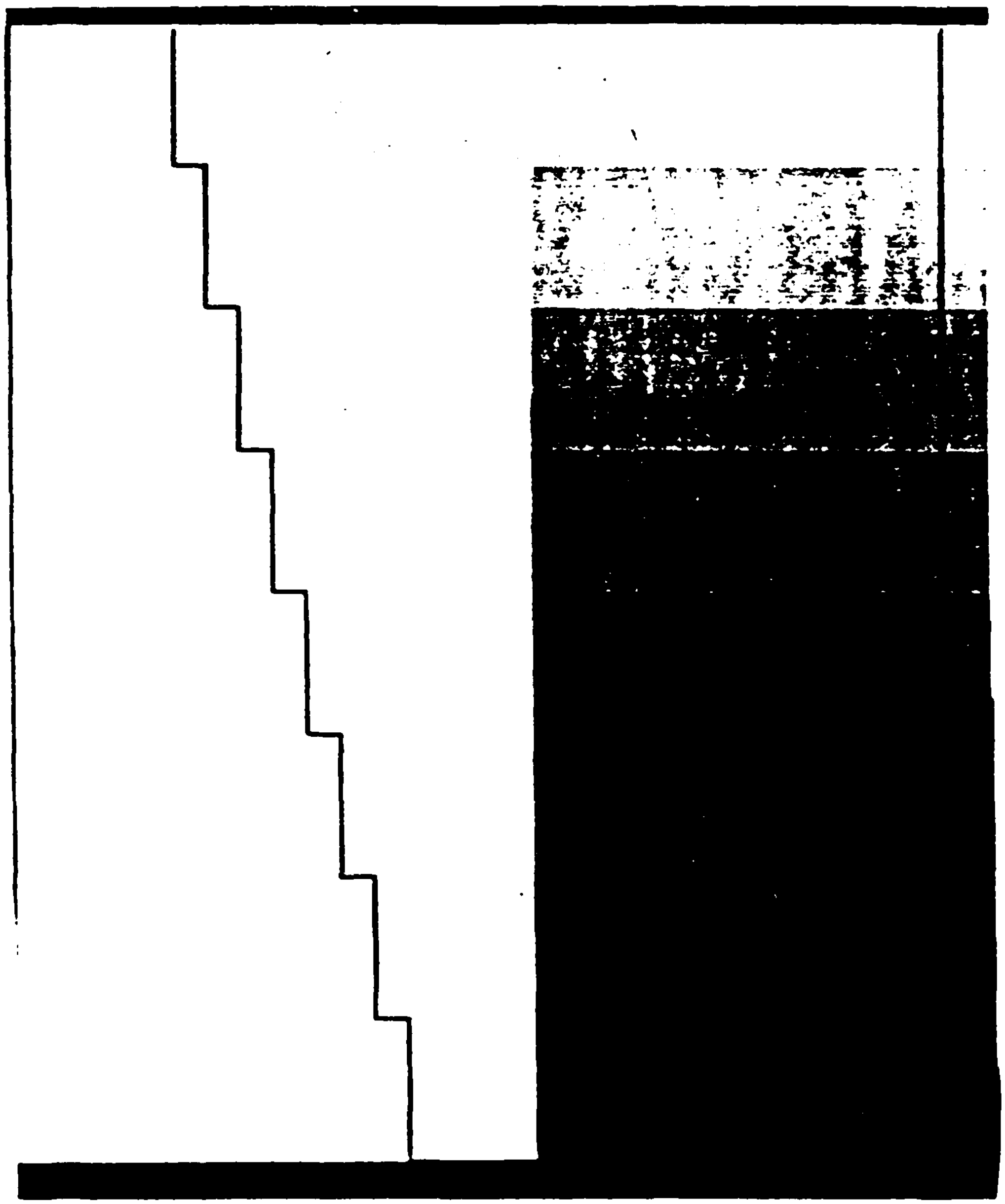


Figure E-1. Original Scene

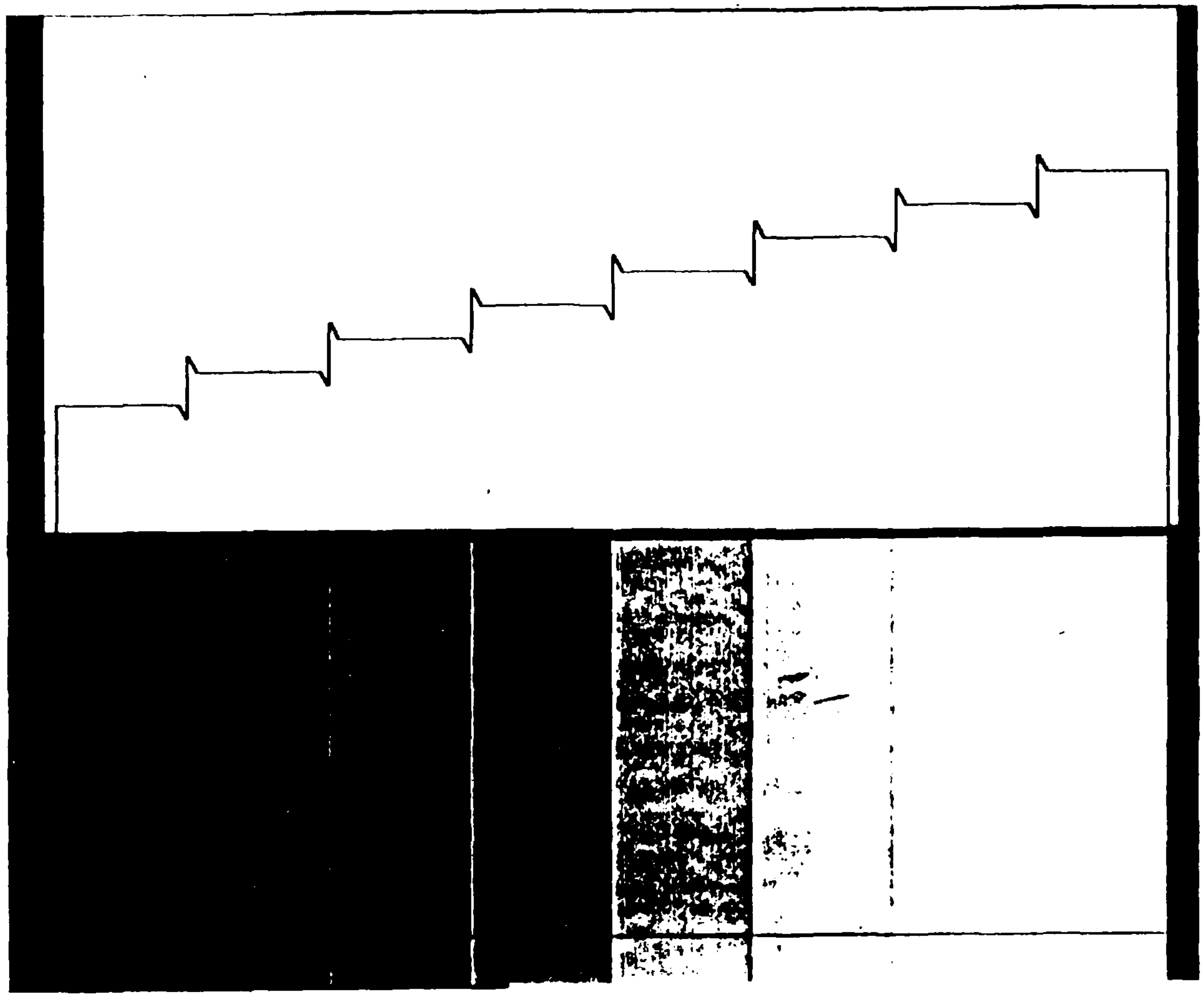


Figure E-2. Scene Modified By [pix = pix + diff]

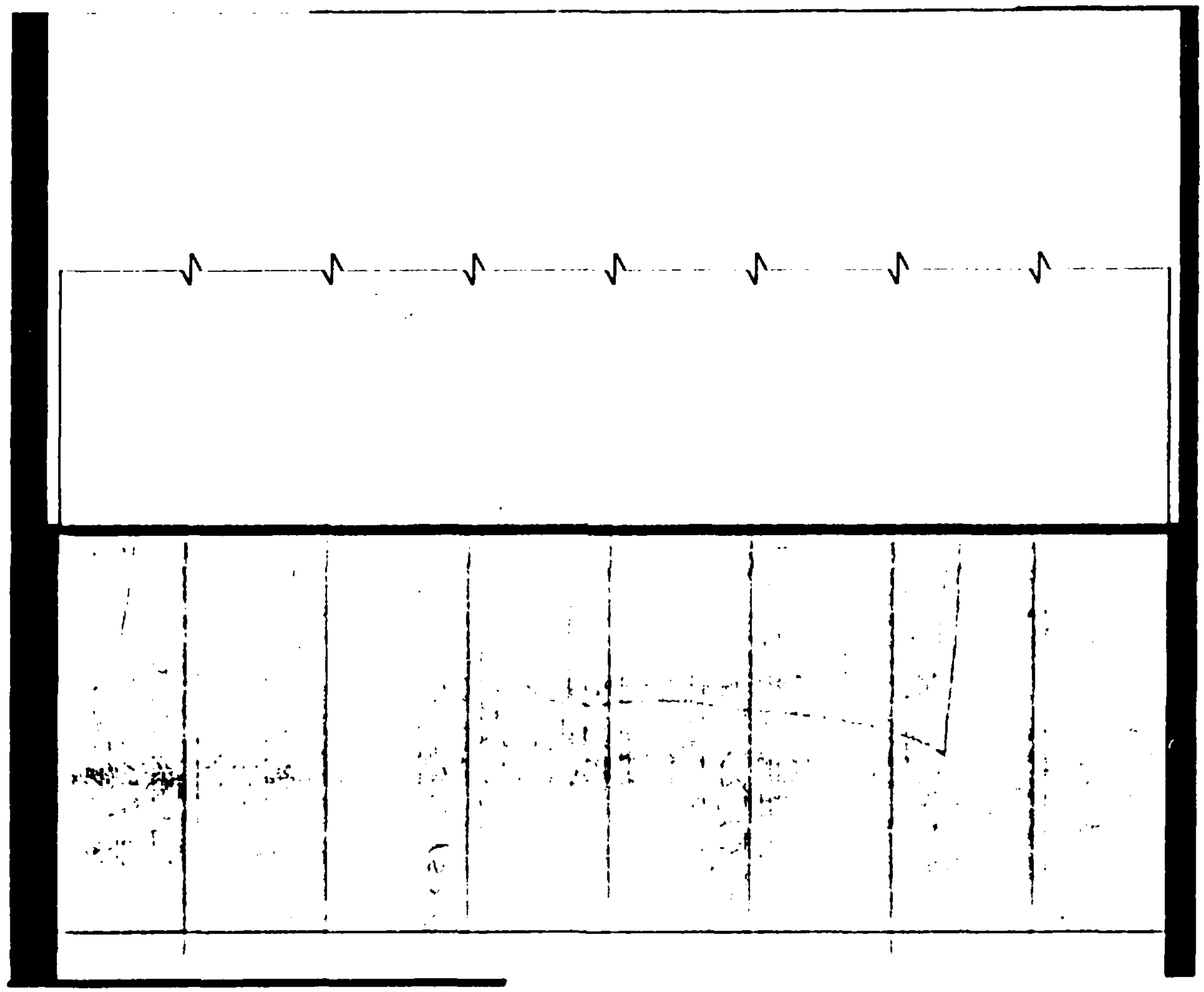


Figure E-3. Scene Modified By [pix = 128 + diff]

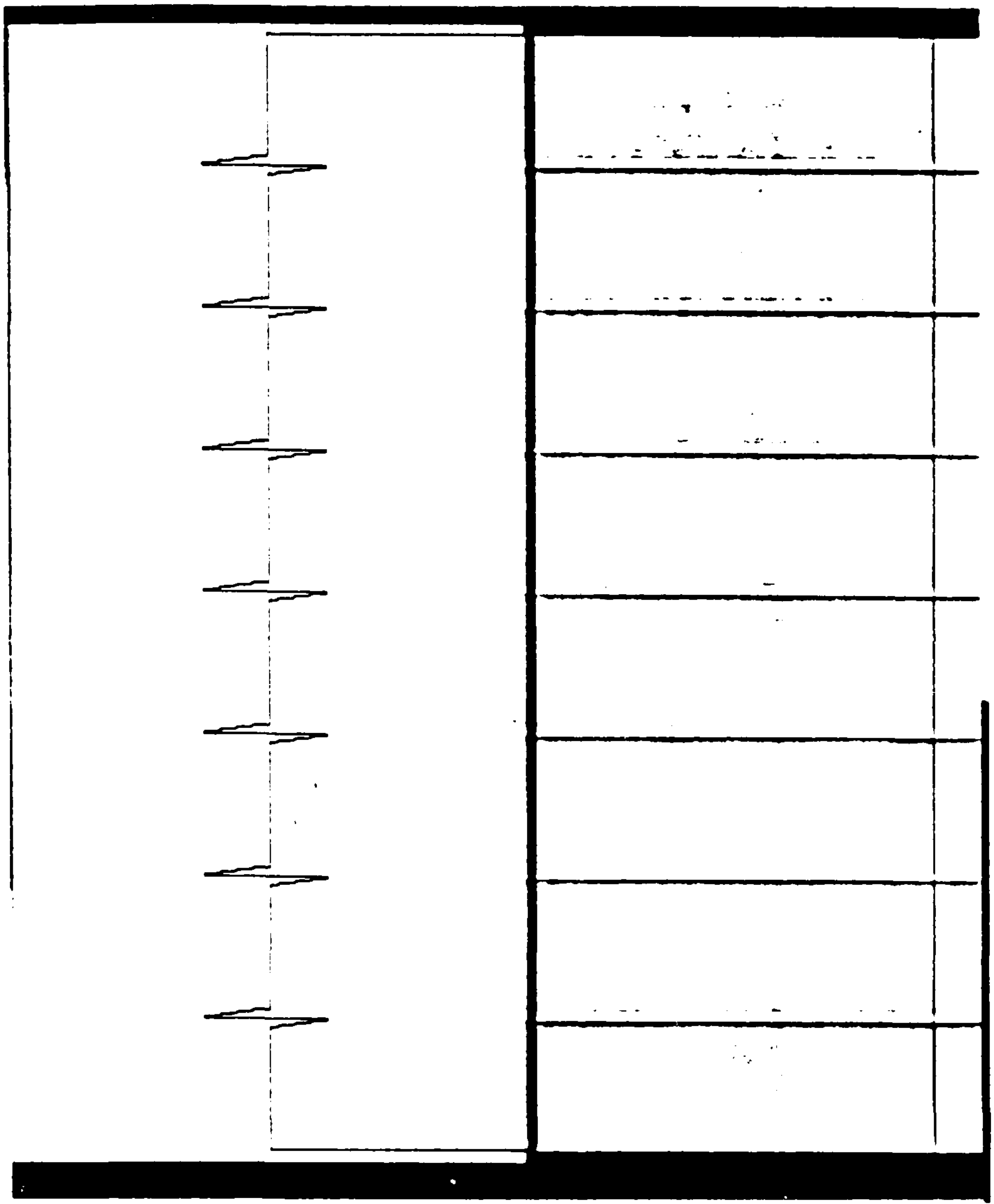


Figure E-4. Scene Modified By [pix = 128 + 4*diff]

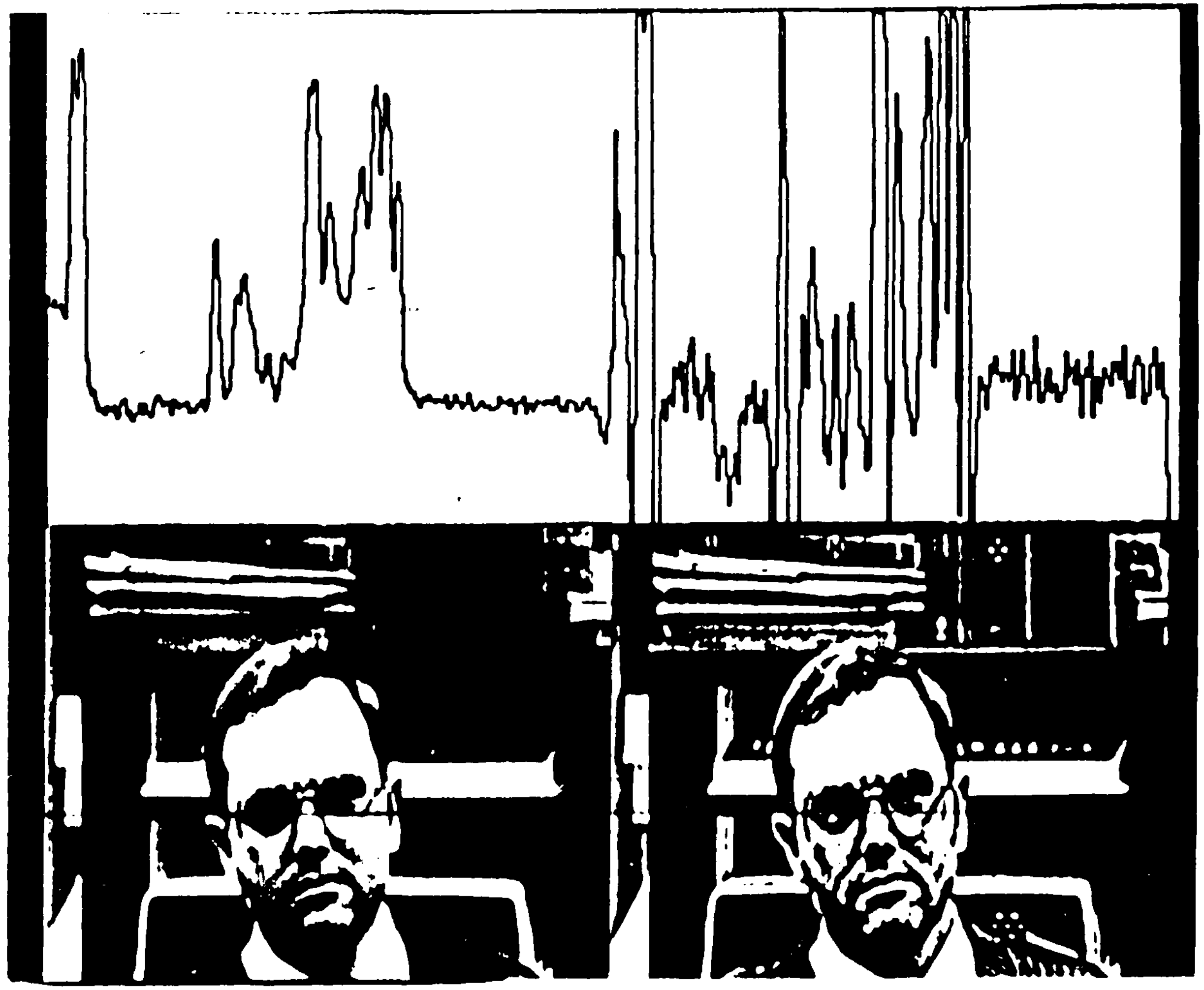


Figure E-5. Scene Modified By [pix = pix + 3*diff]

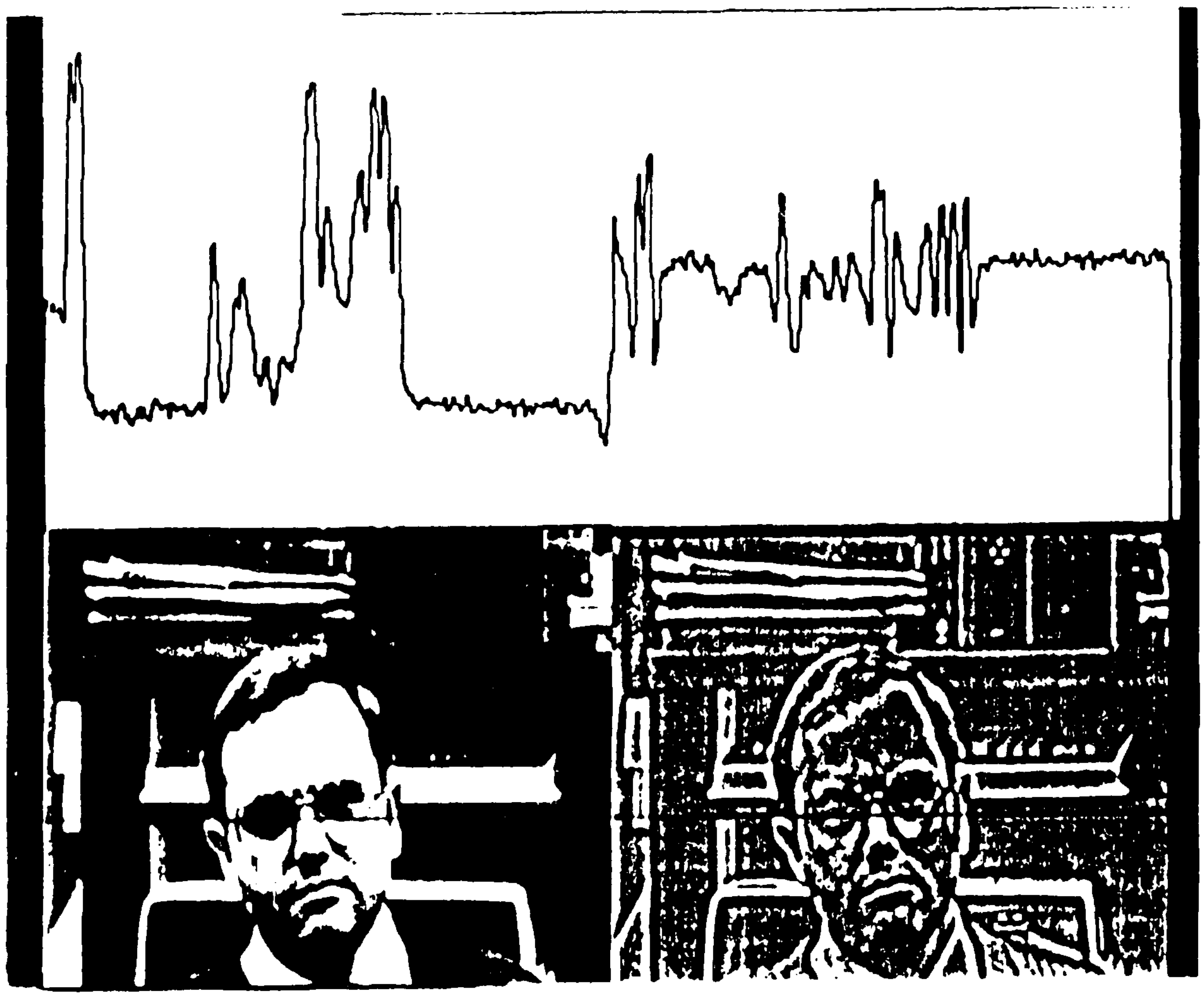


Figure E-6. Scene Modified By [pix = 128 + diff]

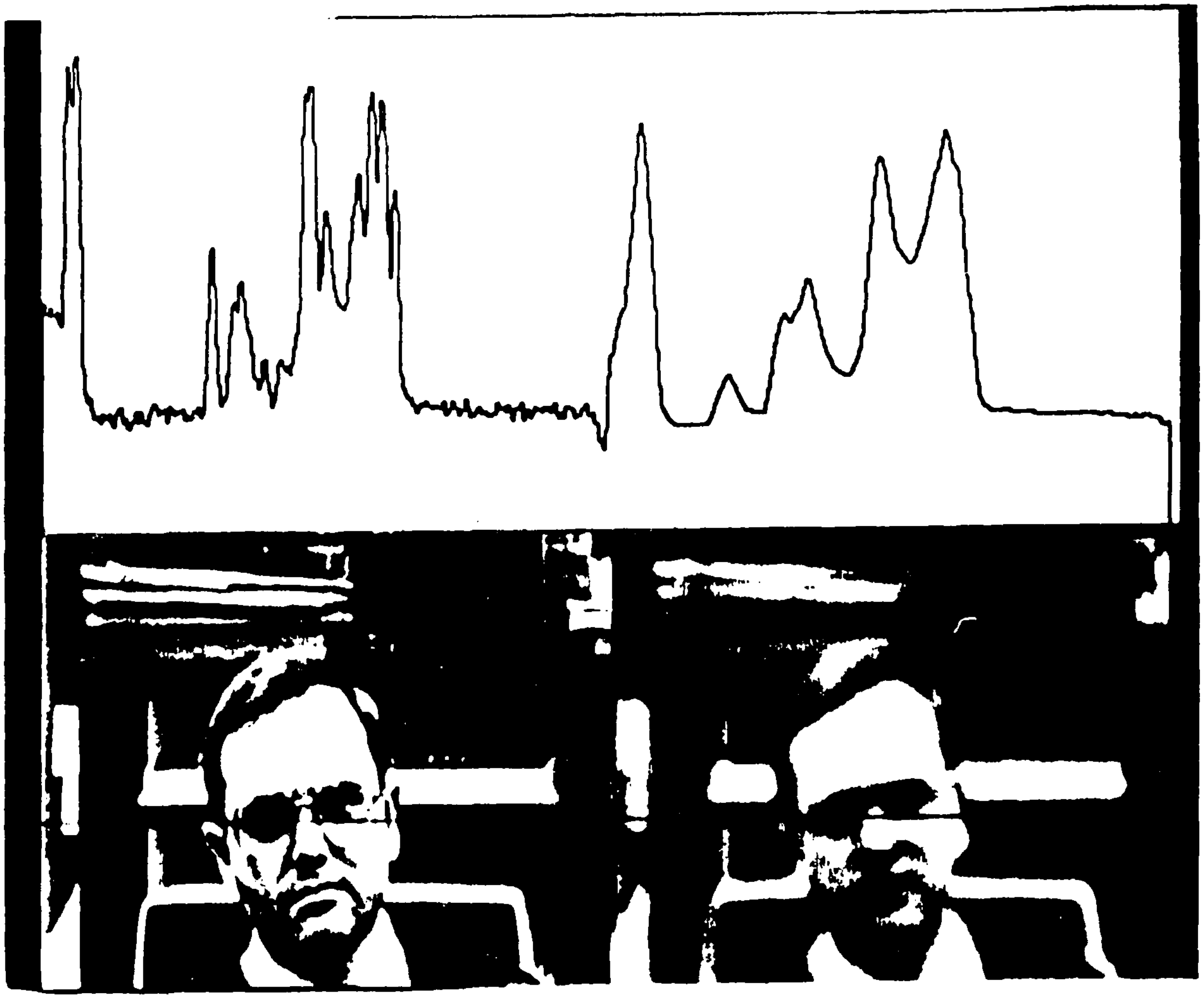


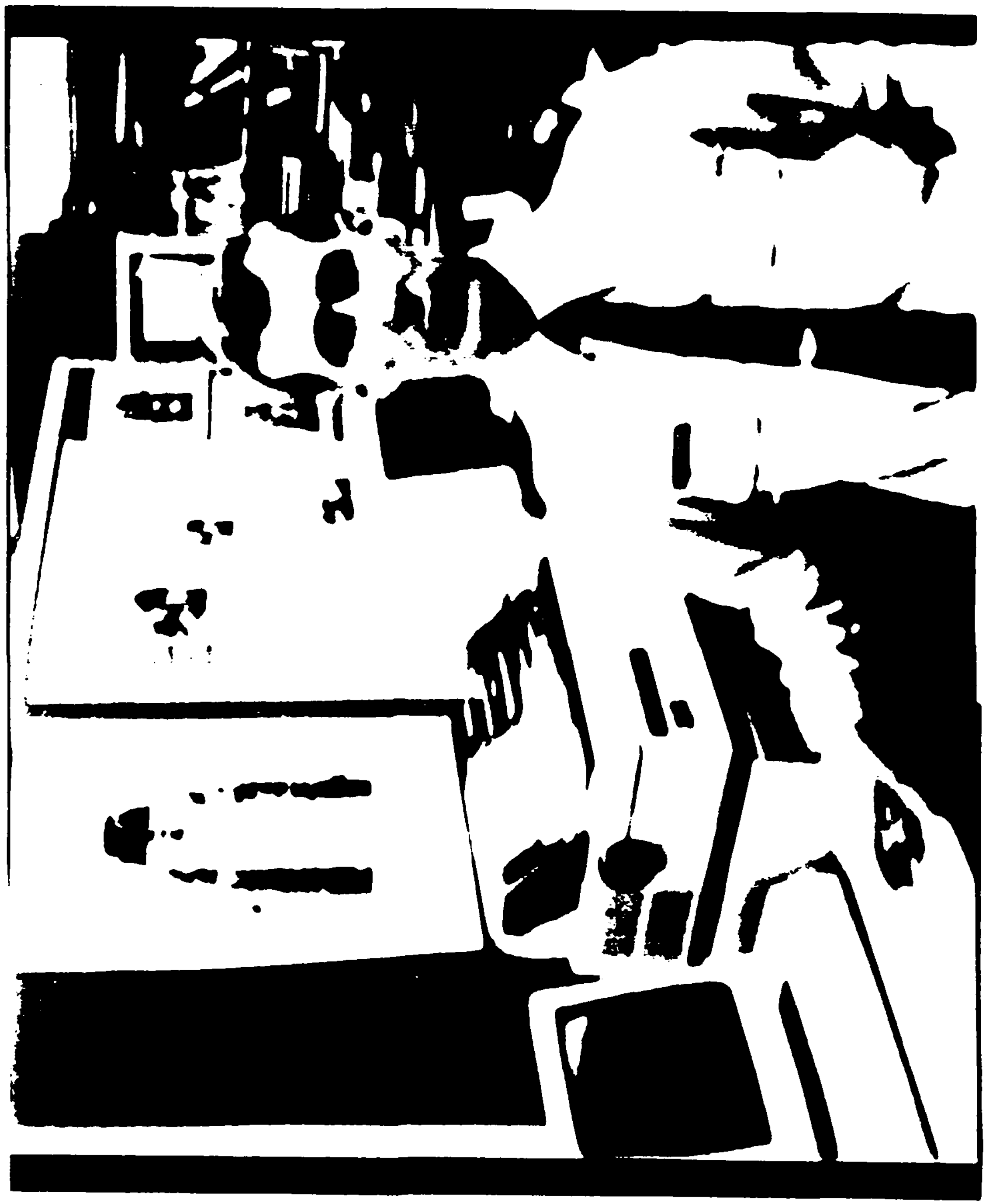
Figure E-7. Scene Modified By [pix = avg]

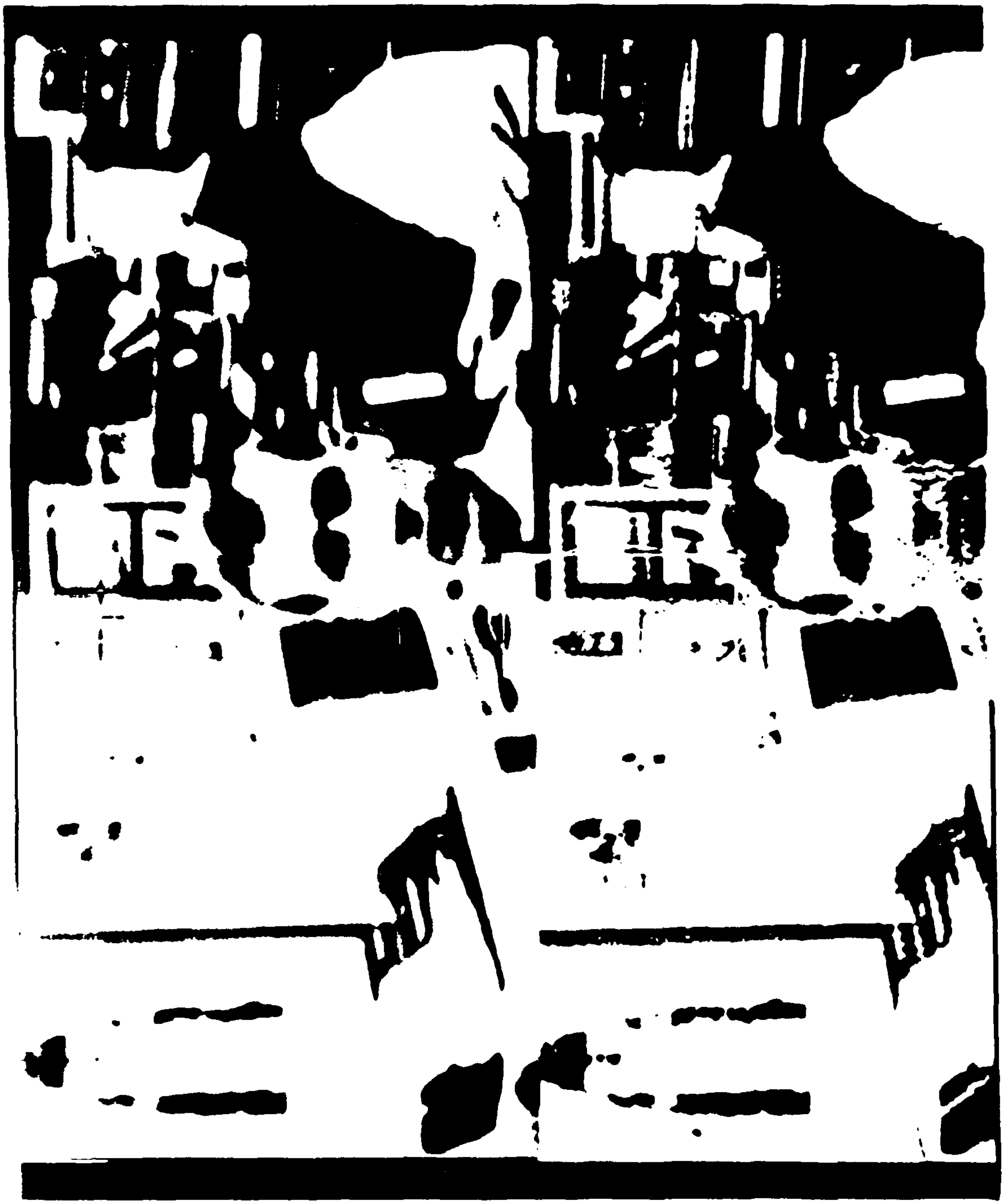
Appendix F

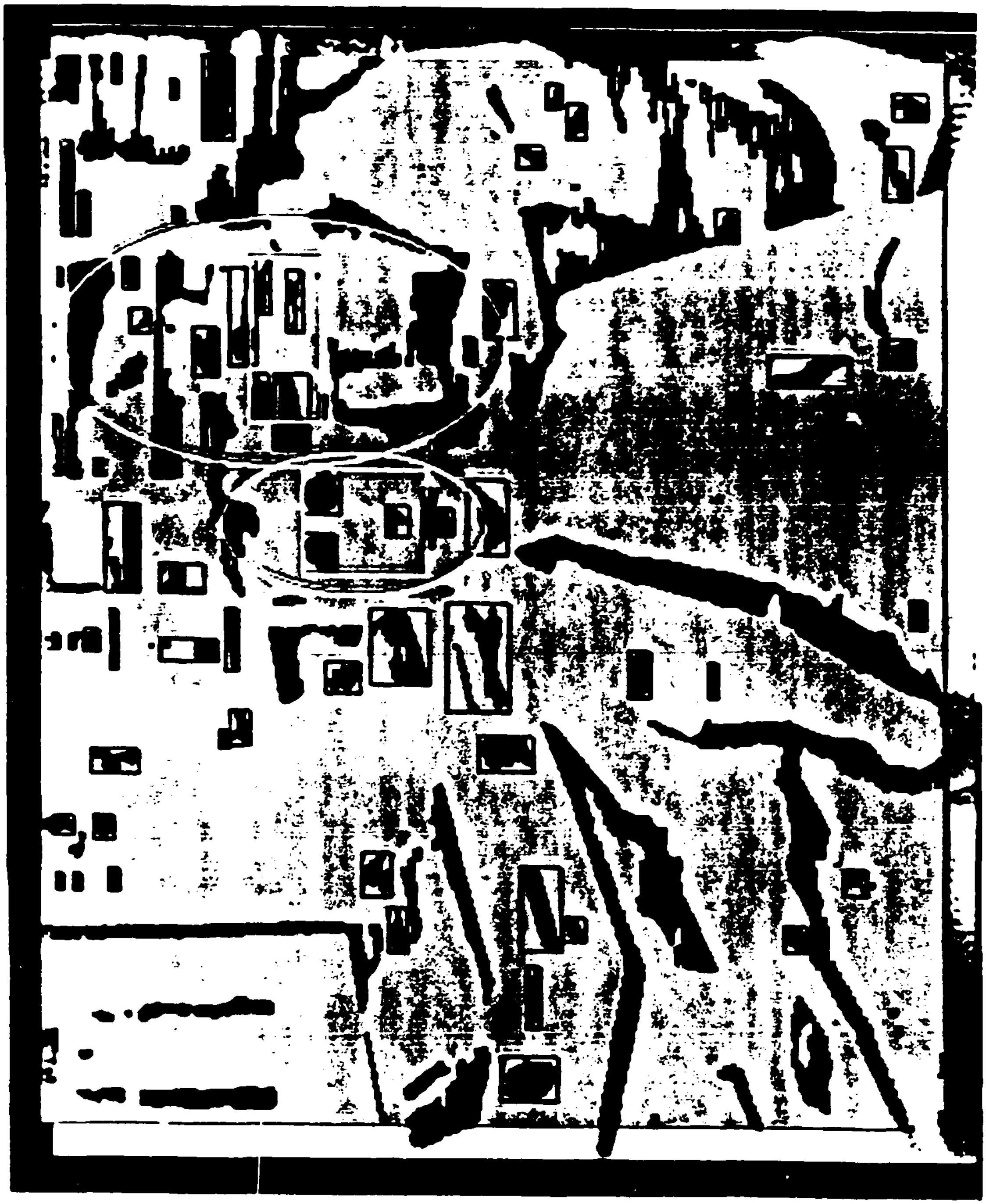
Scenes Used to Test Face Location

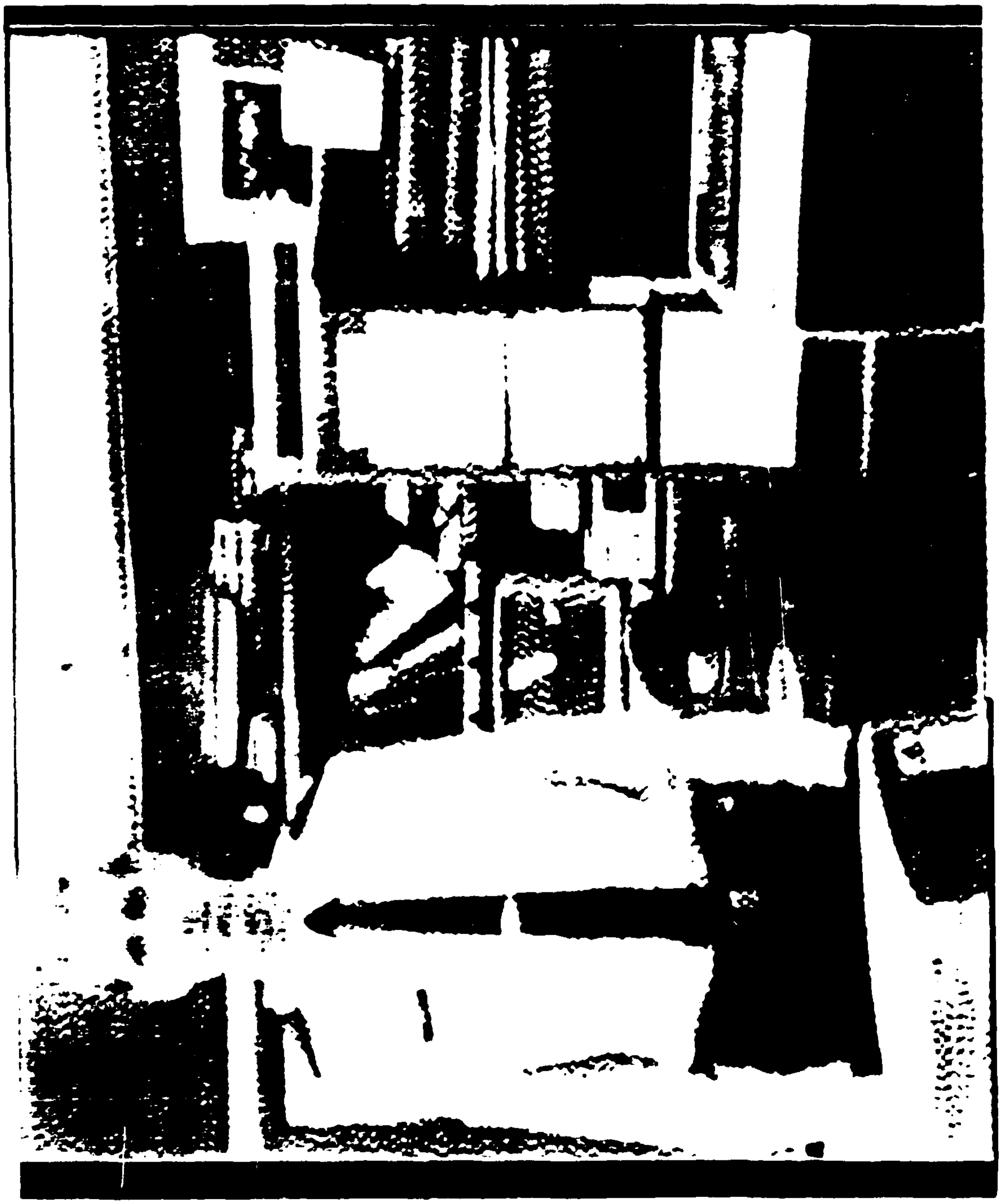
The scene numbers shown below correspond to the numbers in Table 5-2. If a false alarm was found in a scene, the false alarm is shown on the page following that scene. The top of the false alarm scene shows the location of the facial signature found, and the bottom of the scene shows the false face isolated by an ellipse.

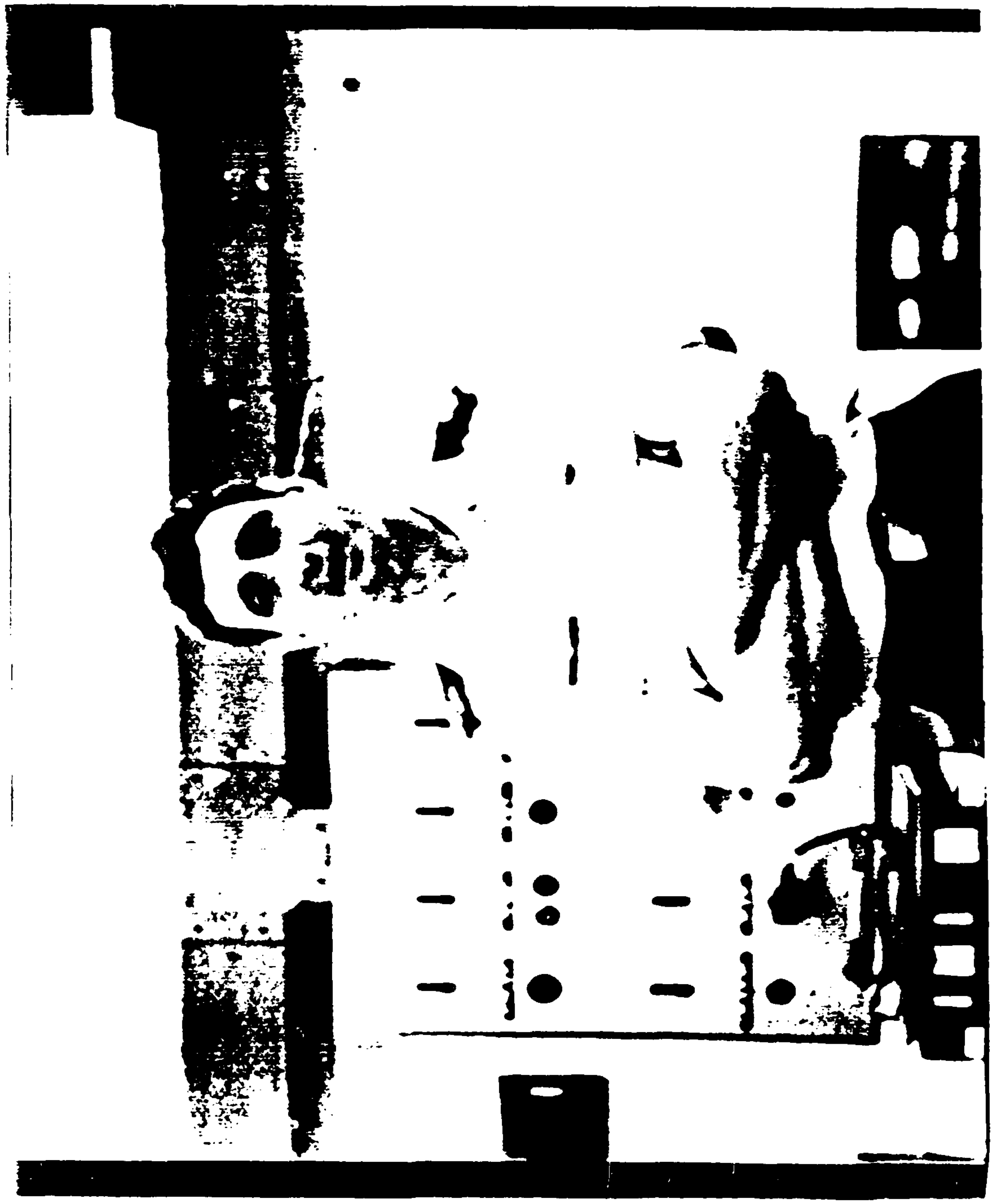
Page	Scene #	Page	Scene #
F-2	1	F-15	11
F-4	2	F-16	12
F-7	3	F-17	13
F-8	4	F-19	14
F-9	5	F-20	15
F-10	6	F-21	16
F-11	7	F-22	17
F-12	8	F-25	18
F-13	9	F-27	19
F-14	10	F-28	20



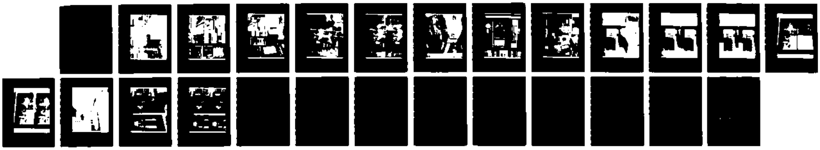


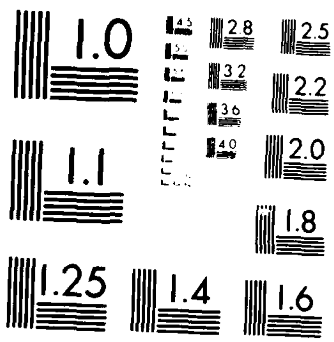




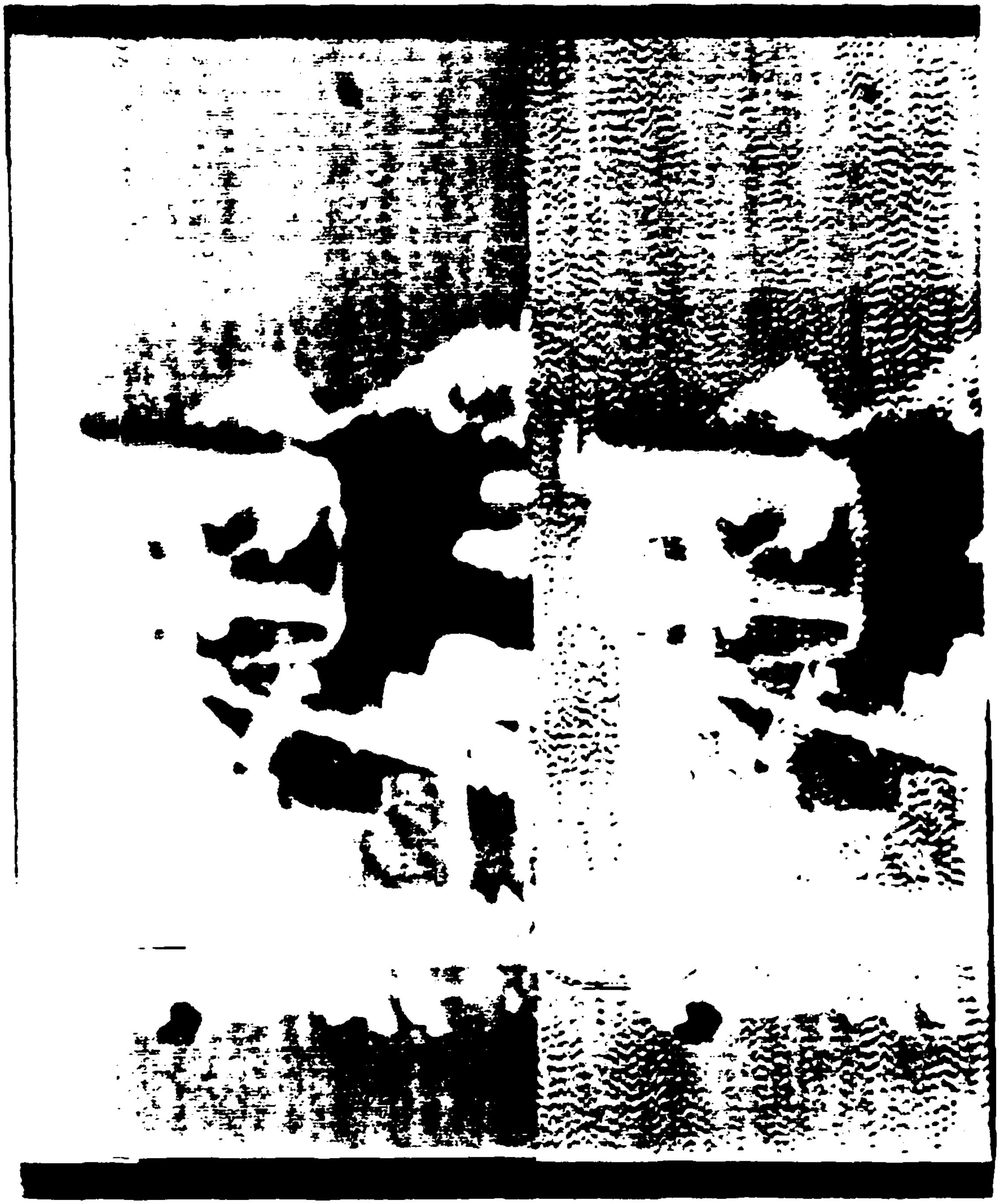


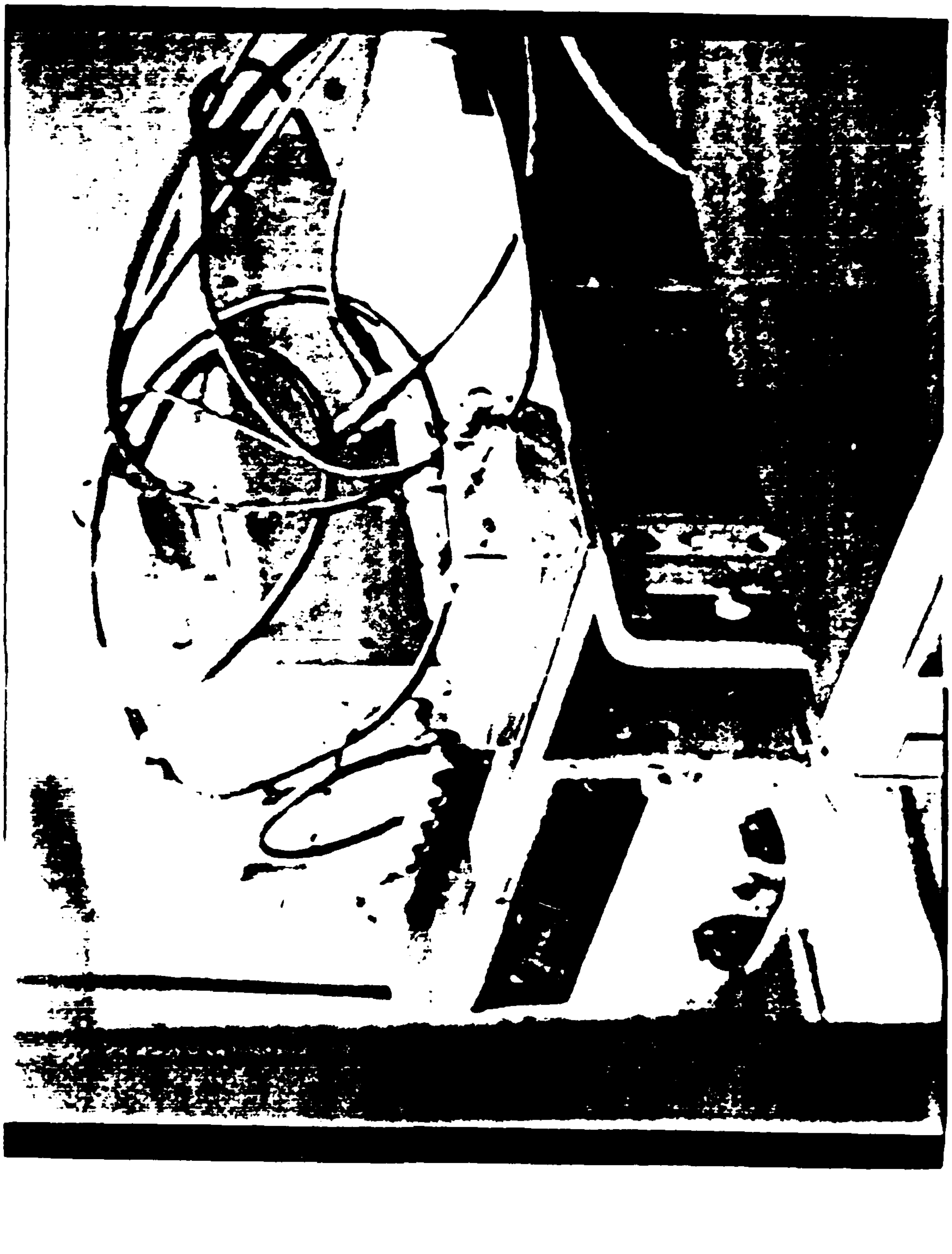
AD-A188 819 EVALUATION AND ENHANCEMENT OF THE AFIT AUTONOMOUS FACE 3/3
RECOGNITION MACHINE(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
UNCLASSIFIED L C LAMBERT DEC 87 AFIT/GE/ENG/87D-35 F/G 12/9 NL

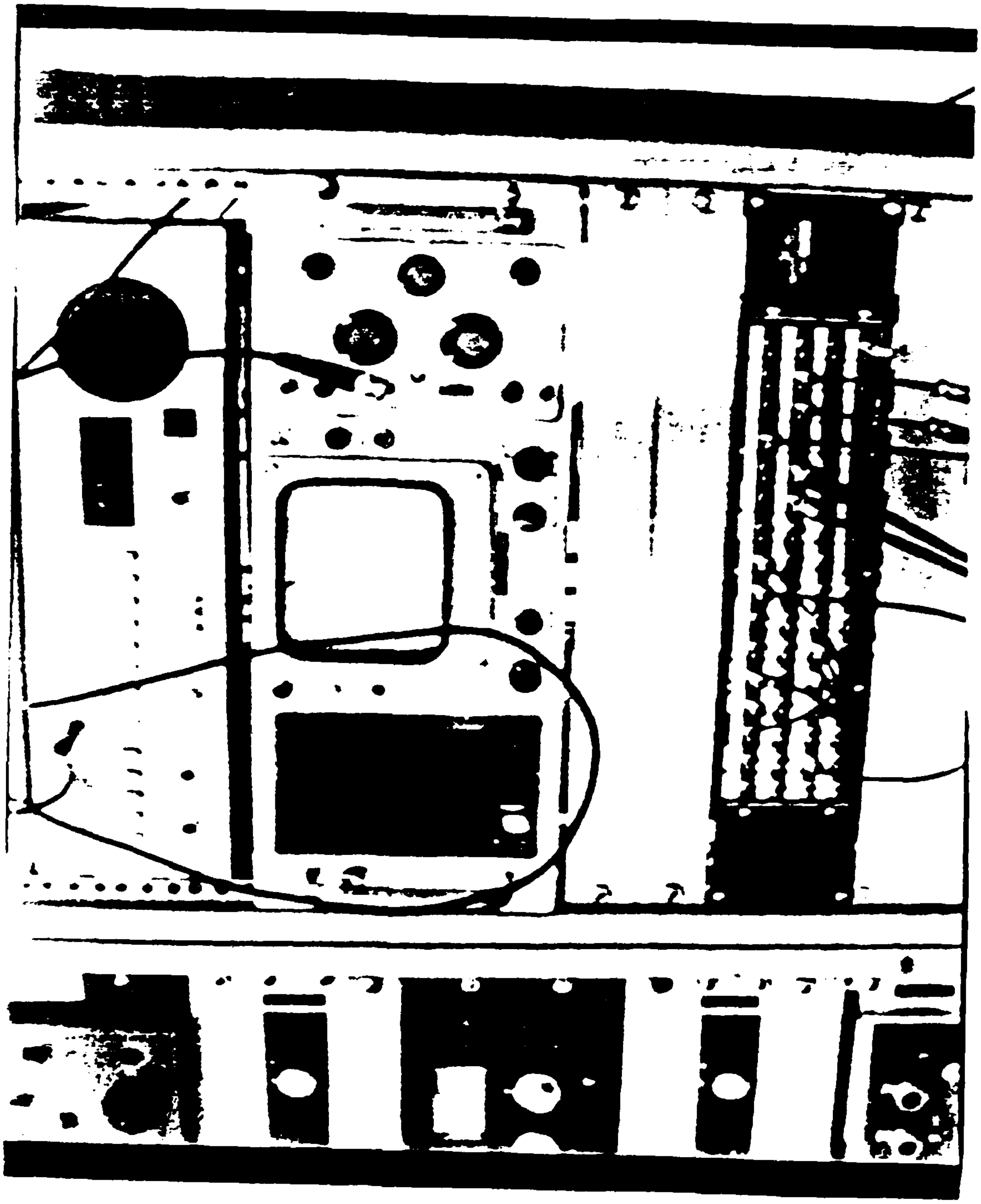


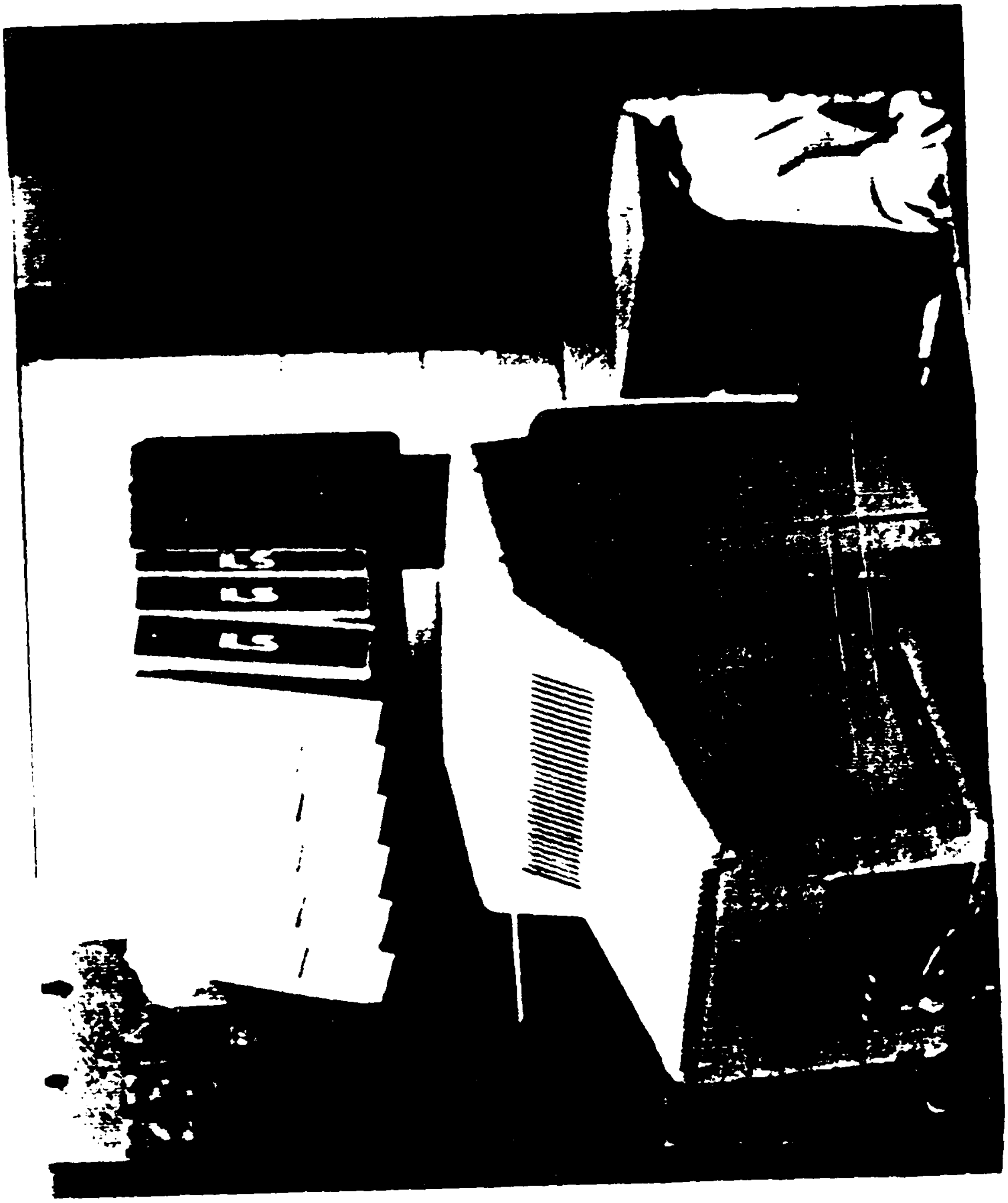


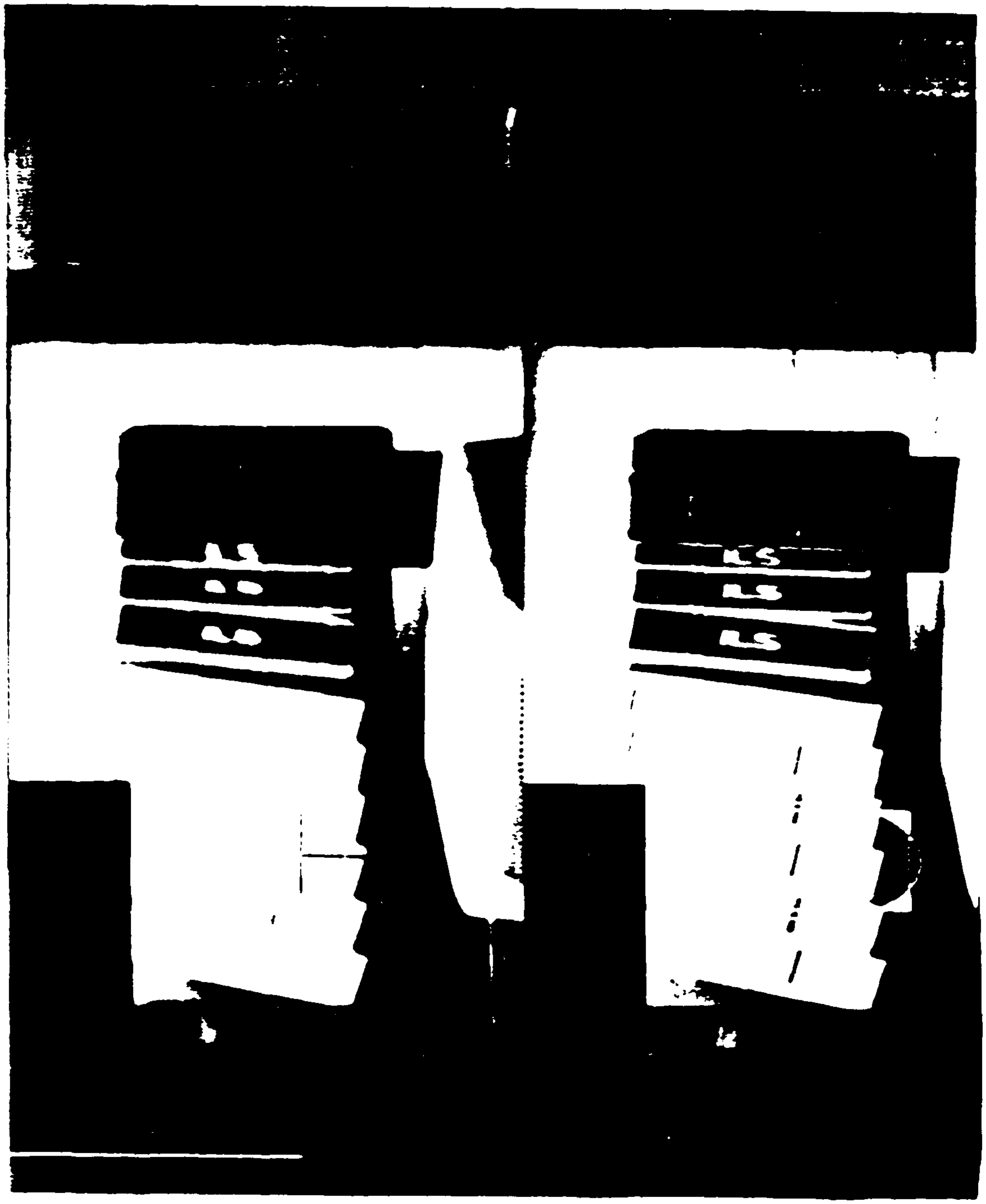
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

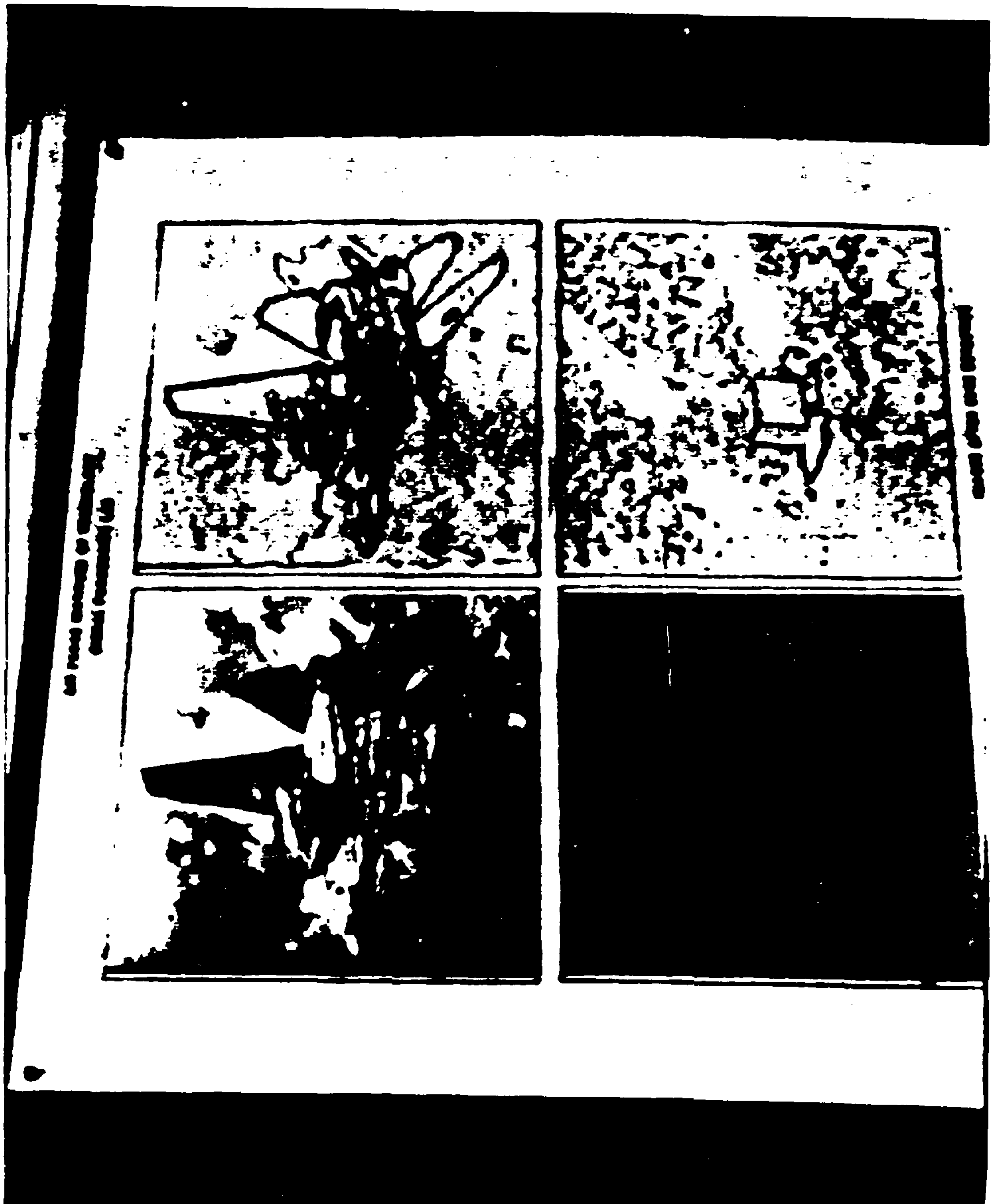


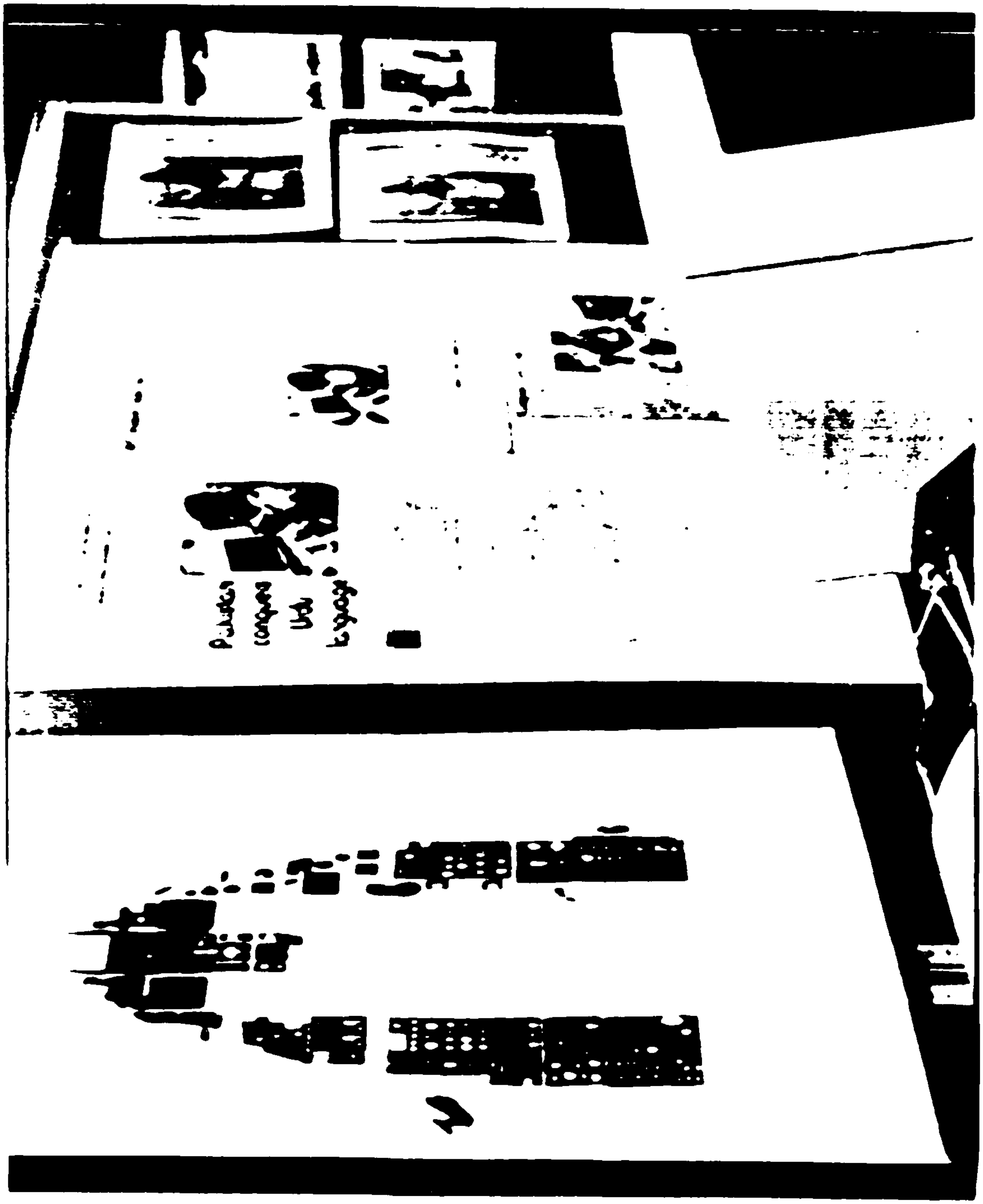


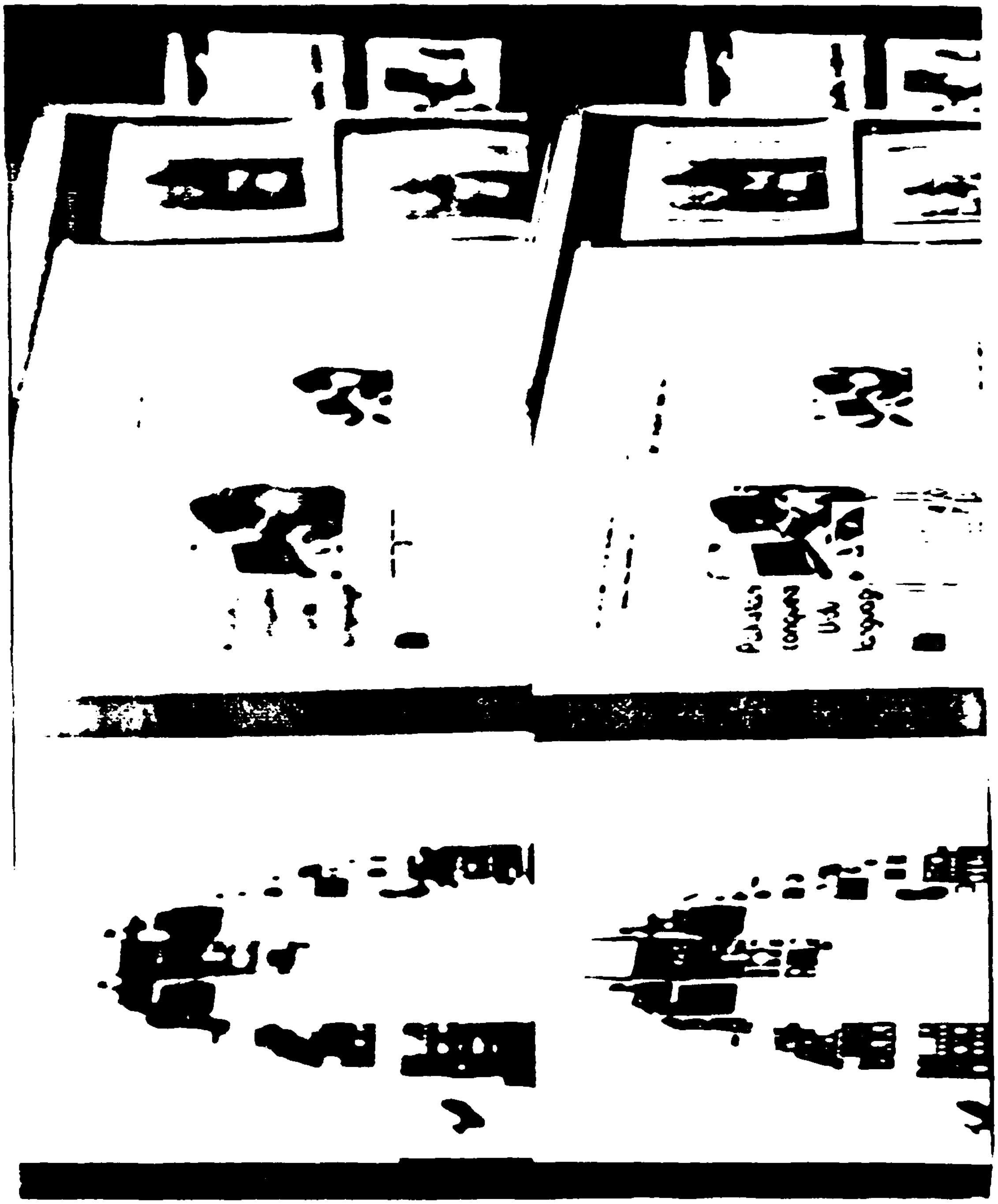





LS
LS
LS
LS
LS
LS







Appendix G

Fast Gestalt Calculation

One of the recommendations for further research with the AFRM is to make it process multiple pictures of a subject's face. In order to do this, the AFRM has to be made as fast as possible. Figure 4-2 shows that the longest time spent in the processing loop is 3 minutes for gestalting the face. If this time can be reduced significantly, then a real-time processing of faces will be possible.

Recognizing that the gestalt calculation is basically a center-of-mass calculation, a much simpler and faster algorithm can be performed. This calculation is shown in the code on page G-4. The new calculation was tested and was found to give nearly the same results in only 5 seconds. The difference in results is due to the fact that the calculation treats all points as constant mass, but some points may have different values (the contrast enhancement doesn't produce a purely binary scene).

Gestalt values for a face were obtained using both calculations and are shown below.

	Old 3 minute Gestalt	New 5 second Gestalt
1	27,55	27,55
2	48,59	46,59
3	38,40	38,38
4	38,61	38,61
5	27,59	27,59
6	40,99	40,99

The two sets of gestalt data yeilded slightly different distances in recognition, but these differences did not alter the recognition results (ordering of candidates). Still, it would be wise to maintain a database where all gestalt values are obtained from the same source.

The new gestalt calculation is implemented as follows:

1. A copy of FACE.C was made and called NEWFACE.C.
2. CORTRAN16 is replaced by the subroutine on page G-4.
3. RTRANSA and RTRANSB are deleted.
4. The source code is compiled and linked in accordance with the User's Manual.
5. The executable program, NEWFACE.EXE, is placed in the directory [FACE].
6. The LOGIN.COM file for account "FACE" is modified, changing the line "run face" to "run newface".

To go back to the original version, simply delete the program NEWFACE.EXE and reverse the change made in step 6 above.

```
/*******************************************************/
cortran16()
{
     int j,i,iwinmax,xtot,ytot,num;
     double c,bmax,ir3d,jr3d;
xtot = ytot = num = 0;
for (i=1; i<iy+2; i++) {
 for (j=1; j<ix+2; j++) {
  if (cray[j][i] > 100) {
   xtot += j;
   ytot += i;
   num++;
}}}
ir3d = (double) (ytot/num);
jr3d = (double) (xtot/num);;
iwinmax = iy;                                /* scale */
if (ix > iwinmax) iwinmax = ix;
ir3d3 = ir3d*(128.0/(double)iwinmax) + 0.5;
jr3d3 = jr3d*(128.0/(double)iwinmax) + 0.5;
return;
}
/*******************************************************/
```

Bibliography

Bromley, L. K. Computer-Aided Processing Techniques For Useage in Real-Time Image Evaluation. Masters Thesis, University of Houston, May 1977.

Edwards, Betty, Drawing on the Right Side of the Brain. L.A. California: J.P. Turcher Publishing, 1979.

Goldstein, Alvin G.; Mackenberg, Edmund J. "Recognition of Human Faces from Isolated Facial Features: A Developmental Study", Psychonomic Science, Vol 6, No 4, 1966.

Haith, Marshall M.; Bergman, Terry and Moore, Michael J. "Eye Contact and Face Scanning in Early Infancy", Science, Vol 198, 25 Nov 1977.

ITEX-100 Programmer's Manual. Part # 47-S10008-02. Imaging Technology Inc. Woburn MA, 1986.

Kabrisky, Matthew, Director Signal Processing Laboratory. Personal Interview. Air Force Institute of Technology Wright-Patterson AFB OH. May 1987.

Kernighan, Brian W.; Ritchie, Dennis M. The C Programming Language. Bell Laboratories. Prentice Hall, 1978.

Luria, A. L. Human Brain and Psychological Processes. New York N.Y. Harper & Row Publishers, 1966.

Routh, Richard L. Cortical Thought Theory: A Working Model of the Human Gestalt Mechanism. PhD Dissertation, AFIT/DS/EE/85-1, Air Force Institute of Technology, DTIC Document, July 1985.

Russel, Robert l. Personal Interviews. Air Force Institute of Technology, August and November 1987.

-----. Performance of a Face Recognition Machine Using Cortical Thought Theory. Masters Thesis, AFIT/GE/ENG/85D, Air Force Institute of Technology, DTIC Document, December 1985.

Smith, Edward J. Development of an Autonomous Face Recognition Machine. Masters Thesis, AFIT/GE/ENG/86D-36, Air Force Institute of Technology, DTIC Document, December 1986.

Werblin, Frank S. "The Study of Sensitivity in the Retina", Scientific American, Jan 1973.

VITA

Captain Laurence C. Lambert was born on 29 October 1960 in Pittsfield, Massachussettes. He attended the University of Lowell, Lowell Massachussettes, and received the degree of Bachelor of Science in Electrical Engineering in May 1982. After working for a year at the University of Lowell Research Foundation, he entered the USAF Officer Training School and was commissioned in June 1983. He then served as a Systems Integration Engineer for the Life Support Systems Programming Office, Aeronautical Systems Division at Wright-Patterson AFB until entering the School of Engineering, Air Force Institute of Technology, in June 1986.

Permanent address: 44 Lexington Ave #4
Magnolia, Mass 01924

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b RESTRICTIVE MARKINGS:

2a SECURITY CLASSIFICATION AUTHORITY:

2b DECLASSIFICATION / DOWNGRADING SCHEDULE:

3 DISTRIBUTION / AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

4 PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/87D-35

5 MONITORING ORGANIZATION REPORT NUMBER(S):

6a NAME OF PERFORMING ORGANIZATION: School of Engineering

6b OFFICE SYMBOL *(If applicable)*: AFIT/ENG

7a NAME OF MONITORING ORGANIZATION:

6c ADDRESS *(City, State, and ZIP Code)*: Air Force Institute of Technology, Wright-Patterson AFB, OH 45433

7b ADDRESS *(City, State, and ZIP Code)*:

8a NAME OF FUNDING / SPONSORING ORGANIZATION:

8b OFFICE SYMBOL *(If applicable)*:

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

8c ADDRESS *(City, State, and ZIP Code)*:

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE *(Include Security Classification)*: Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine

12 PERSONAL AUTHOR(S): Lambert, Laurence C. Captain USAF

13a TYPE OF REPORT: MS Thesis

13b TIME COVERED: FROM ______ TO ______

14 DATE OF REPORT *(Year, Month, Day)*: 1987 December

15 PAGE COUNT: 215

16 SUPPLEMENTARY NOTATION:

17 COSATI CODES

FIELD	GROUP	SUB GROUP
09	02	

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*: Computers, Artificial Intelligence, Image Processing, Image Segmentation

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Thesis Chairman: Matthew Kabrisky, PhD
Professor of Electrical Engineering

31 Dec 87

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. Matthew Kabrisky Professor, GS-15

22b TELEPHONE *(Include Area Code)*: (513) 255-5276

22c OFFICE SYMBOL: AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Continued from block 19: Abstract

This thesis evaluates and improves the Autonomous Face Recognition Machine (AFRM) created in 1985 at AFIT. This effort involved re-writing the AFRM code in the C programming language and hosting it on a Micro-VAX II. In addition, several new algorithms were added to the AFRM including: brightness normalization of input images, moving target detection, and a new face location algorithm. The results of this effort include: improved face location, higher recognition accuracy, and near real-time processing.

This thesis includes a complete description of the AFRM and its development history.

END
FILMED
MARCH, 1988
DTIC